basic module

Modules

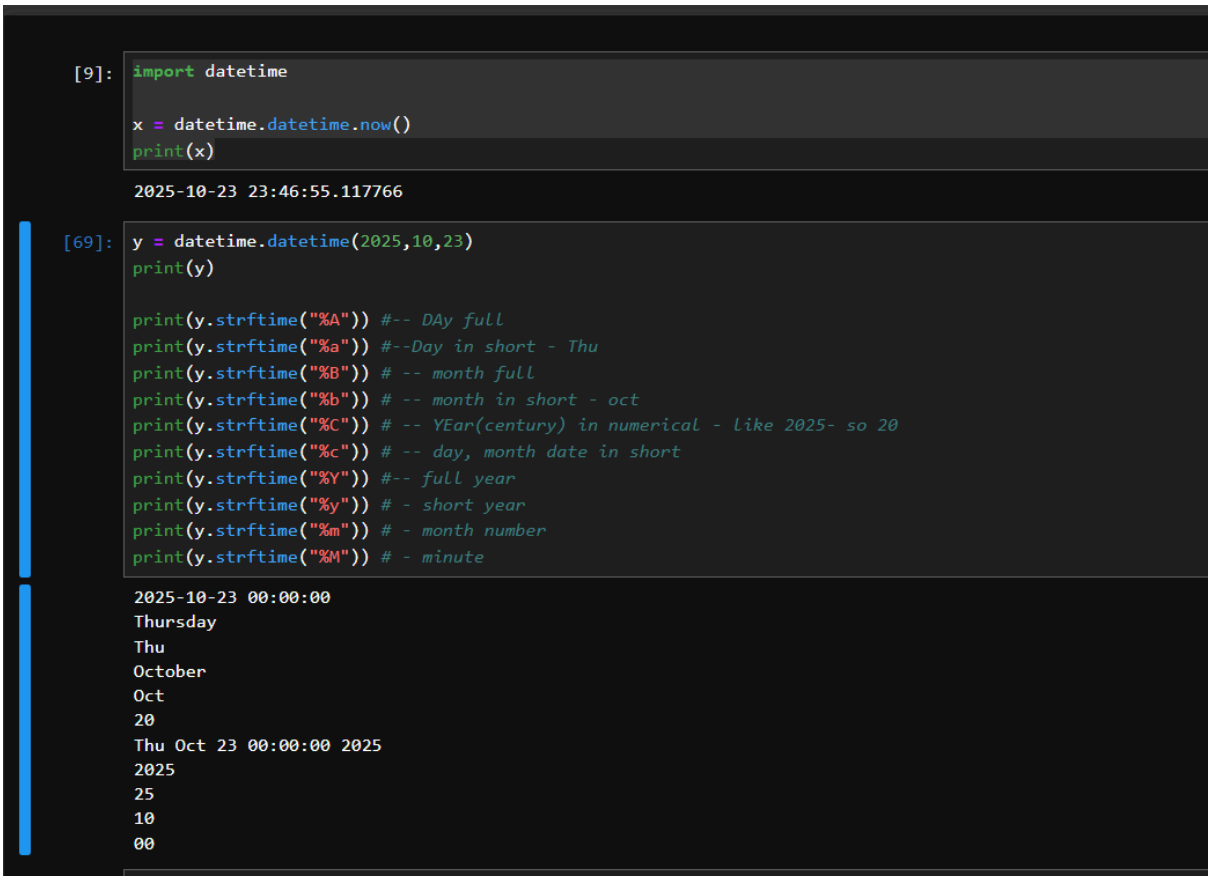
1) Basic Modules -

Module are the (.py) files, that contains set of functions you want to include in your program.

Two types of Modules -

1) Built in Modules (already present in python)
2) Created by us - Modules.

2) Date time Module -



```
[9]: import datetime

     x = datetime.datetime.now()
     print(x)

     2025-10-23 23:46:55.117766

[69]: y = datetime.datetime(2025,10,23)
      print(y)

      print(y.strftime("%A")) #-- DAy full
      print(y.strftime("%a")) #--Day in short - Thu
      print(y.strftime("%B")) # -- month full
      print(y.strftime("%b")) # -- month in short - oct
      print(y.strftime("%C")) # -- YEar(century) in numerical - like 2025- so 20
      print(y.strftime("%c")) # -- day, month date in short
      print(y.strftime("%Y")) #-- full year
      print(y.strftime("%y")) # - short year
      print(y.strftime("%m")) # - month number
      print(y.strftime("%M")) # - minute

      2025-10-23 00:00:00
      Thursday
      Thu
      October
      Oct
      20
      Thu Oct 23 00:00:00 2025
      2025
      25
      10
      00
```

3) Random Module -

```
[79]: import random

      x = random.randint(1,5)
      print(x)

      l = ["Head","Tails"]
      z = random.choice(l)
      print(z)

      4
      Head
```

4) Math Module -

5)

6)

🔢 Numpy

What you will learn?

What is NumPy?

What is NumPy Array ?

NumPy vs Python List

Installation & Import NumPy

importance of numpy in python

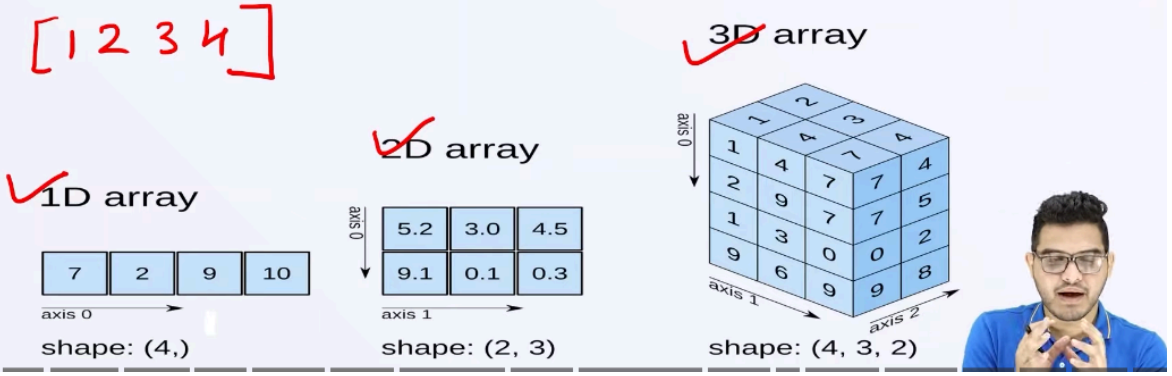
What is NumPy?

- NumPy is the fundamental package for scientific computing in Python.
- NumPy is a Python library that provides a multidimensional array object, various derived objects

What is NumPy Array ?

- An array is a grid of values and it contains information about the raw data, how to locate an element, and how to interpret an element.

[1 2 3 4]



NumPy vs Python List

- **Advantages of using Numpy Arrays Over Python Lists:**
  - consumes less memory.
  - fast as compared to the python List.
  - convenient to use.

Importance of NumPy in Python

WsCube Tec

- wide variety of mathematical operations on arrays.
- It supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.
- mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Difference Between NumPy Array and List in Python

- Data types storage
- Importing module
- Numerical operation
- Modification capabilities
- Consumes less memory
- Fast as compared to the python list
- Convenient to use

% timeit → oneline

%% timeit → Program

Here is proof like which take less time–

```
In [1]: %timeit [j**4 for j in range(1,9)]
        2.55 µs ± 61.1 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

In [2]: import numpy as np
        %timeit np.arange(1,9)**4
        2.02 µs ± 142 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

In [ ]:
```

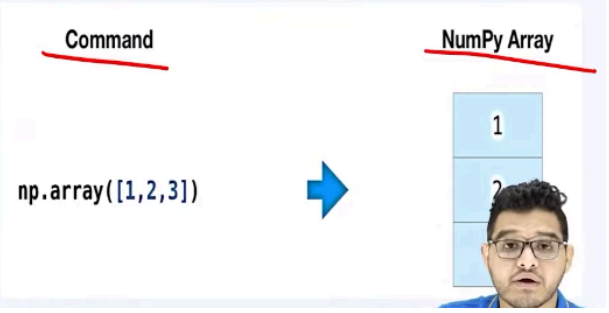
create arrays

Creating NumPy Arrays

WsCube

- To create a NumPy array, you can use the function np.array().

```
import numpy as np
a = np.array( [1, 2, 3] )
print( a )
```



Eg 1 =

```
l = [3,42,59]

a = np.array(l)
print(type(l))
print(a)
print(type(a))
```

Output =

```
<class 'list'>
[ 3 42 59]
<class 'numpy.ndarray'>
```

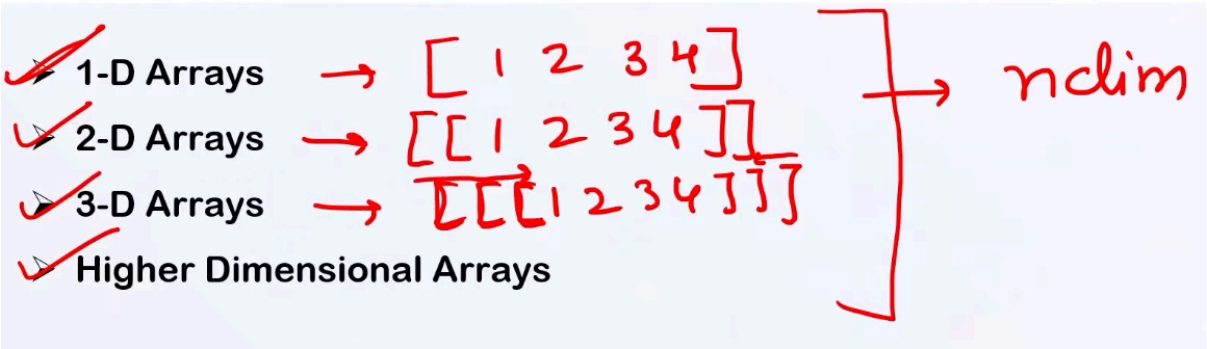
Eg 2 =

```
l = []

for i in range(1,5):
    x = int(input("Enter digits : "))
    l.append(x)

print(np.array(l))
```

Output = `[43 56 23  4]`

Now Dimensions in Arrays

Dimensions in Arrays



1) How to check dimensions in arrays.

By = ndim

Eg 1-

```
a = np.array([3,42,59])

print(a)
print(type(a))
print(a.ndim) #-- it will tell
```

```
Output -
[ 3 42 59]
<class 'numpy.ndarray'>
1
```

Eg 2 - (2d)

```
ar2 = np.array([[11,23,12],[32,53,1]])
print(ar2)
print(ar2.ndim)
```

```
Output -
[[11 23 12]
 [32 53  1]]
2
```

Eg 3 - (3d)

There may be error if i don't keep same no of digits in both, all elements.

```
ar3 = np.array([[[11,23,12],[32,1],[42,93,99]]])
print(ar3)
print(ar3.ndim)
```

[15] 0.0s

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[15], line 1
----> 1 ar3 = np.array([[[11,23,12],[32,1],[42,93,99]]])
      2 print(ar3)
      3 print(ar3.ndim)

ValueError: setting an array element with a sequence. The requested array has an inhomogeneous shape after 2 dimensions.
```

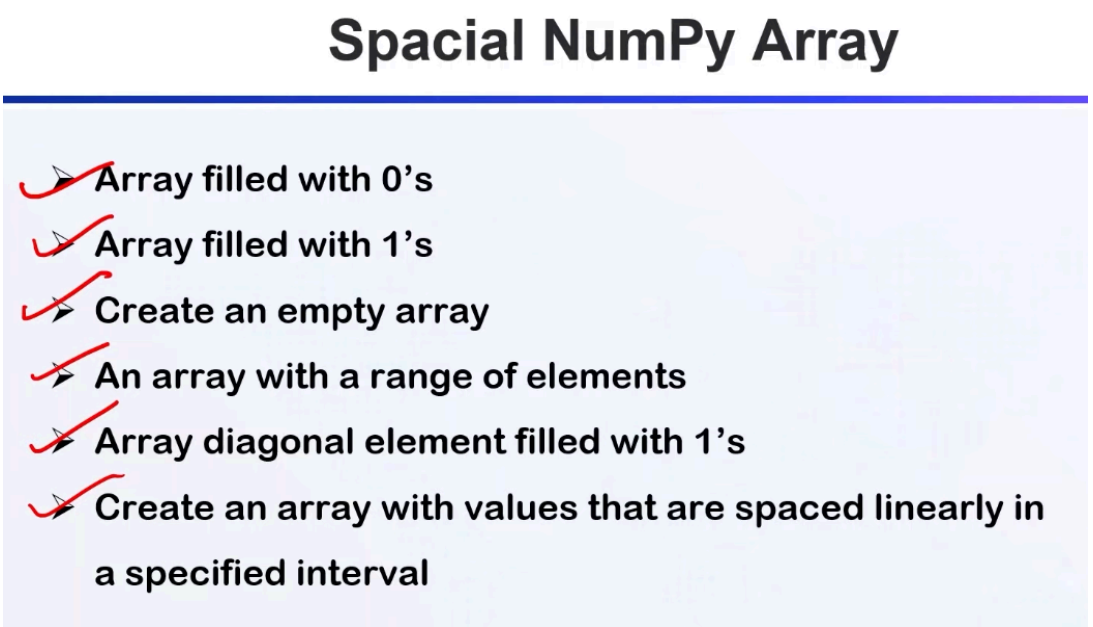
ValueError: setting an array element with a sequence. The requested array has an inhomogeneous shape after 2 dimensions. The detected shape was (1, 3) + inhomogeneous part.

Eg 4 - (of n no of dimensions)

```
arn = np.array([23,42,3],ndmin=10)
print(arn)
print(arn.ndim)
```

```
Output =
[[[[[[[[[[23 42  3]]]]]]]]]]
10
```

2)


Spacial NumPy Array
Array filled with 0's
Array filled with 1's
Create an empty array
An array with a range of elements
Array diagonal element filled with 1's
Create an array with values that are spaced linearly in a specified interval

1) 0's

```
arr_zero = np.zeros(4)
print(arr_zero)
print(arr_zero.ndim)
print(" \n")

Output -
[0. 0. 0. 0.]
1

arr_zero1 = np.zeros((3,4))
print(arr_zero1)
print(arr_zero1.ndim)

Output -
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
2

print(" \n")
arr_zero2 = np.zeros([1,3,3,6]) #as much i will dimen.
will incr.
print(arr_zero2)
print(arr_zero2.ndim)

Output -
[[[[0. 0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0. 0.]]

  [[0. 0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0. 0.]]

  [[0. 0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0. 0.]]]]
4
```

2) 1's

```
ar_one = np.ones(5)
print(ar_one)
print(ar_one.ndim)

ar_one1 = np.ones((4,2,3))
print(ar_one1)
print(ar_one1.ndim)
```

Output =

```
[1. 1. 1. 1. 1.]
1

[[[1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]]]
3
```

3) Empty arrays

But it stores or shows.. Previous arrays digits…

```
Eg -
ar_emp = np.empty(5)
print(ar_emp)
print(ar_emp.ndim)

Output -
[1. 1. 1. 1. 1.]
1
```

4) Range

```
Eg -
ar_rn = np.arange(5)
print(ar_rn)
print(ar_rn.ndim)
```

```
Output -
[0 1 2 3 4]
1
```

5) Diagonal

Eg -

```
ar_di = np.eye(3)
print(ar_di)

Output -
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

ar_di1 = np.eye(3,5)
print(ar_di1)

Output -
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]]
```

6) Linespace – (its means how many parts it will divide equally) like (for 20 in 3 parts so, 0 to 10, 10, to 20)

Eg

```
ar_lin = np.linspace(0,20,num= 3)
print(ar_lin)

Output -
[ 0. 10. 20.]
```

random, data_type

Create NumPy Arrays with Random Numbers

WsCube Tech

- **rand( ) :** the function is used to generate a random value between 0 to 1.
- **randn( ) :** the function is used to generate a random value close to zero . This may return positive or negative numbers as well.
- **ranf( ) :** the function for doing random sampling in numpy. It returns an array of specified shape and fills it with random floats in the half-open interval [0.0, 1.0).
- **randint( ) :** the fuction is used to generate a random number be en a given range. min , Max

Eg- not need to write import random when, you already imported Numpy

```
var = np.random.rand(4)
print(var)
```

[61] ✓ 0.0s

```
[0.67397306 0.42091096 0.46662685 0.3684271 ]
```

```
var1 = np.random.rand(2,4)
print(var1)
print(var1.ndim)
```

[69] ✓ 0.0s

```
[[0.46942542 0.73872277 0.32841502 0.92717411]
 [0.88989789 0.76184868 0.52145481 0.38549895]]
2
```

```
var2 = np.random.randn(3)
print(var2)
```

[70] ✓ 0.0s

```
[ 0.46918654 -0.420276    0.21252196]
```

```
var3 = np.random.ranf(4)
print(var3) ## - ranf -- [0.0 to 1.0) means [= include 0.0, and ) = exclude 1
```

[72] ✓ 0.0s

```
[0.37406055 0.81090875 0.02745911 0.44497919]
```

```
var4 = np.random.randint(1,5) #(min,max,no of elements)
print(var4)
```

```
var5 = np.random.randint(1,5,3)
print(var5)
```

Output -

```
4
[4 4 1]
```

DATA TYPES IN NUMPY ARRAYS

Data Types NumPy Arrays

WsCube T

Sr.No.	Data Types	Description
1	bool_	Boolean (True or False) stored as a byte
2	int_	Default integer type (same as C long; normally either int64 or int32)
3	Intc	Identical to C int (normally int32 or int64)
4	Intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
5	int8	Byte (-128 to 127)
6	int16	Integer (-32768 to 32767)
7	int32	Integer (-2147483648 to 2147483647)
8	int64	Integer (-9223372036854775808 to 9223372036854775807)
9	uint8	Unsigned integer (0 to 255)

Sr.No.	Data Types	Description
10	uint16	Unsigned integer (0 to 65535)
11	uint32	Unsigned integer (0 to 4294967295)
12	uint64	Unsigned integer (0 to 18446744073709551615)
13	float_	Shorthand for float64
14	float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
15	float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
16	float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
17	complex_	Shorthand for complex128

Sr.No.	Data Types	Description
18	complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
19	complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Data Types as Function

list of characters that are used to represent dtype in Numpy:

i -	integer
b -	boolean
u -	unsigned integer
f -	float
c -	complex float
m -	timedelta
M -	datetime
O -	object
S -	string
U -	Unicode string
V -	the fixed chunk of memory for other types (void)

Eg -

```
[3]
import numpy as np

var = np.array([2,42,321,4])

print("data type : ",var.dtype)
```

✓ 0.2s

```
data type :  int32
```

```
[8]
var1 = np.array([3.23,5.2,43.94,42])

print(var1.dtype)
```

✓ 0.0s

```
float64
```

```
[10]
var = np.array(["af",'a',"kfa"])

print(var.dtype)
```

✓ 0.0s

```
<U3
```

```
[11]
var = np.array([234,4,"fak","KJFSL"])

print(var.dtype)
```

✓ 0.0s

```
<U11
```

Now it's time - to change data type

```
x = np.array([2,34,54,3],dtype = np.int8)

print(x.dtype)
print(x)
```

✓ 0.0s

```
int8
[ 2 34 54  3]
```

```
x1 = np.array([2,34,54,3],dtype = "f")

print(x1.dtype)
print(x1)
```

✓ 0.0s

```
float32
[ 2. 34. 54.  3.]
```

```
x2 = np.array([2,34,54,3])

new = np.float32(x2)

print(x2.dtype)
print(x2)
print(new.dtype)
print(new)
```

✓ 0.0s

```
int32
[ 2 34 54  3]
float32
[ 2. 34. 54.  3.]
```

shape & Reshape

Shape & Reshape

1) **Check shape — like**
   In 2 X 2 Matrix = 2 rows, 2 column

```
import numpy as np

var = np.array([[23,42,3],[32,55,2]])

print(var)
print()
print(var.shape)
print(var.ndim)
```

Output -

```
[[23 42  3]
 [32 55  2]]

(2, 3)
2
```

2) **2nd way…**

```
var1 = np.array([1,24,53,2], ndmin= 4)

print(var1)
print(var1.ndim)
print(var1.shape)
```

Output -

```
[[[[ 1 24 53  2]]]]
4
(1, 1, 1, 4)
```

RESHAPE

3) **2 D**

```
import numpy as np

var2 = np.array([2,4,3,5,6,7,9,8,10])
```

```
print(var2)
print(var2.ndim)

print()
x = var2.reshape(3,3)
print(x)
print(x.ndim)
```

Output -

```
[[ 2  4  3  5  6  7  9  8 10]]
2

[[ 2  4  3]
 [ 5  6  7]
 [ 9  8 10]]
2
```

4) 3 D

```
var3 = np.array([2,4,3,5,6,7,9,8,10,11,12,1])
print(var3)
print(var3.ndim)

print()
xz = var3.reshape(2,3,2)
print(xz)
print(xz.ndim)
```

Output -

```
[ 2  4  3  5  6  7  9  8 10 11 12  1]
1
************************
[[[ 2  4]
  [ 3  5]
  [ 6  7]]

 [[ 9  8]
  [10 11]
  [12  1]]]
3
```

5) Multi Dimensional to One Dimensional.

```
var3 = np.array([2,4,3,5,6,7,9,8,10,11,12,1,2,4,3,5,6,7])
print(var3)
```

```
print(var3.ndim)

print()
xz = var3.reshape(3,3,2)
print(xz)
print(xz.ndim)

print("*******")
one = xz.reshape(-1)
print(one)
print(one.ndim)
```

Output -

```
[ 2  4  3  5  6  7  9  8 10 11 12  1  2  4  3  5  6  7]
1

[[[ 2  4]
  [ 3  5]
  [ 6  7]]

 [[ 9  8]
  [10 11]
  [12  1]]

 [[ 2  4]
  [ 3  5]
  [ 6  7]]]
3
*******
[ 2  4  3  5  6  7  9  8 10 11 12  1  2  4  3  5  6  7]
1
```

6)

7)

Airthmatic operations

Arithmetic Operations

1) Arithmetic operations…

a+b	np.add(a,b)
a-b	np.subtract(a,b)
a*b	np.multiply(a,b)
a/b	np.divide(a,b)
a%b	np.mod(a,b)
a**b	np.power(a,b)
1/a	np.reciprocal(a)

- np.min(x)
- np.max(x)
- np.argmin(x)
- np.sqrt(x)
- np.sin(x)
- np.cos(x)
- np.cumsum(x)

2) 1 D

```
var = np.array([1,3,5,2,4])
x = var - 1
print(x)
```
[30] ✓ 0.0s

```
[0 2 4 1 3]
```

```
var = np.array([1,3,5,2,4])
x = var / 3
print(x)
```
[31] ✓ 0.0s

```
[0.33333333 1.         1.66666667 0.66666667 1.33333333]
```

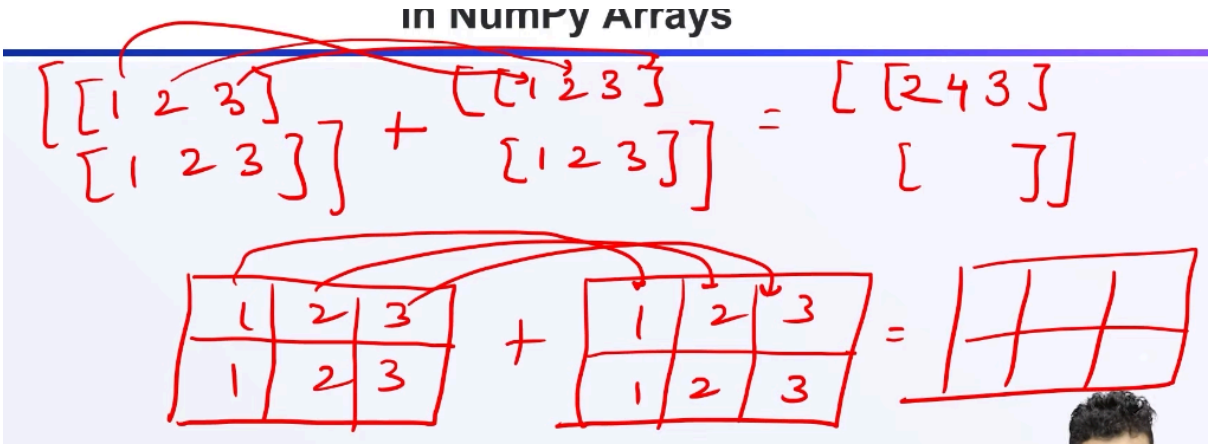
```
var = np.array([1,3,5,2,4])
x = var * 5
print(x)
```
[32] ✓ 0.0s

```
[ 5 15 25 10 20]
```

```
var = np.array([1,3,5,2,4])
var1 = np.array([3,4,5,1,6])

x = var + var1
print(x)
```
[39] ✓ 0.0s

```
[ 4  7 10  3 10]
```

3) 2 D



Eg

```
var21 = np.array([[1,2,3,4],[1,2,3,4]])
var22 = np.array([[1,2,3,4],[1,2,3,4]])
print(var21.ndim,"\n")
x = np.add(var21,var22)
print(var21,"\n")
print(var22,"\n")
print(x)
print(x.ndim)
```

✓ 0.0s

```
2

[[1 2 3 4]
 [1 2 3 4]]

[[1 2 3 4]
 [1 2 3 4]]

[[2 4 6 8]
 [2 4 6 8]]
2
```

```
var21 = np.array([[1,2,3,4],[1,2,3,4]])
var22 = np.array([[1,2,3,4],[1,2,3,4]])
x = np.multiply(var21,var22)

print(var21,"\n")
print(var22,"\n")
print(x)
print(x.ndim)
```

✓ 0.0s

```
[[1 2 3 4]
 [1 2 3 4]]

[[1 2 3 4]
 [1 2 3 4]]

[[ 1  4  9 16]
 [ 1  4  9 16]]
2
```

```
var21 = np.array([[1,2,3,4],[1,2,3,4]])
var22 = np.array([[1,2,3,2],[1,2,3,3]])
x = np.divide(var21,var22)

print(var21,"\n")
print(var22,"\n")
print(x)
print(x.ndim)
```

✓ 0.0s

```
[[1 2 3 4]
 [1 2 3 4]]

[[1 2 3 2]
 [1 2 3 3]]

[[1.         1.         1.         2.        ]
 [1.         1.         1.         1.33333333]]
2
```

4) Another Operations…

- np.min(x)
- np.max(x)
- np.argmin(x)
- np.sqrt(x)
- np.sin(x)
- np.cos(x)
- np.cumsum(x)

np.argmin = tell index no of min or max…

Eg -

1D

```
import numpy as np

var = np.array([23,42,2,5,6,9,10])

print("min : ",np.min(var),"and, ",np.argmin(var))
print("max : ",np.max(var),"and, ",np.argmax(var))
print("sqrt : ",np.sqrt(var))
print("sin : ",np.sin(var))
print("cos : ",np.cos(var))
```

Output -

```
min :  2 and,  2
max :  42 and,  1
sqrt :  [4.79583152 6.4807407  1.41421356 2.23606798
2.44948974 3.
 3.16227766]
sin :  [-0.8462204  -0.91652155  0.90929743 -0.95892427
-0.2794155   0.41211849
 -0.54402111]
```

```
cos :  [-0.53283302 -0.39998531 -0.41614684  0.28366219
0.96017029 -0.91113026
 -0.83907153]
```

2D (Now min and max for --> 2-D)

```
var1 = np.array([[2,3,5],[91,2,8]])

#-- for column --> 0, for Row --> 1...()

print("Min : ",np.min(var1,0))
print("Max : ",np.max(var1,0))
print()
print("Min : ",np.min(var1,1))
print("Max : ",np.max(var1,1))
```

```
Output -
Min :  [2 2 5]
Max :  [91  3  8]

Min :  [2 2]
Max :  [ 5 91]
```

5) ldkjfl

insert data by numpy

Insert Data by Numpy

1️⃣ Install NumPy (if needed)

pip install numpy

2️⃣ Insert the Table using NumPy Array

Step-by-step Code

```
import numpy as np

data = np.array([
    [101, "Rahul", "IT", 45000, "2022-01-15", "Delhi", None],
    [102, "Ankit", "HR", None, "2021-08-10", "Mumbai", None],
    [103, "Neha", "Finance", 52000, None, "Pune", None],
    [104, "Rohan", "IT", 48000, "2023-03-01", None, None],
    [105, "Priya", "Marketing", None, "2020-11-25", "Bangalore", None],
    [None, None, None, None, None, None, None],
    [106, "Aman", "Sales", 40000, "2021-06-18", "Delhi", None],
    [107, "Simran", "HR", 38000, None, "Chandigarh", None],
    [108, "Kunal", "IT", None, "2022-09-05", "Noida", None],
    [109, "Pooja", "Finance", 55000, "2020-02-14", "Mumbai", None],
    [None, None, None, None, None, None, None],
    [110, "Arjun", "Sales", 42000, "2023-01-20", None, None],
    [111, "Sneha", "Marketing", 46000, "2021-04-11", "Jaipur", None],
    [112, "Mohit", "IT", 50000, None, "Gurgaon", None],
    [113, "Nisha", "HR", None, "2019-07-30", "Delhi", None],
    [114, "Akash", "Finance", 60000, "2022-12-01", None, None],
    [115, "Kavya", "Sales", None, None, "Mumbai", None],
    [None, None, None, None, None, None, None],
    [116, "Deepak", "IT", 47000, "2023-05-09", "Pune", None],
    [117, "Ritu", "Marketing", 44000, "2021-10-17", None, None],
    [118, "Saurabh", "HR", None, "2020-03-22", "Noida", None],
    [119, "Isha", "Finance", 53000, None, "Bangalore", None]
], dtype=object)

print(data)
```

✅ This **creates the table** inside a NumPy array
✅ `None` represents **NULL / missing values**
✅ `dtype=object` is **mandatory** for mixed data

3️⃣ Add Column Headers (NumPy way)

NumPy **does not support column names natively**, so we store headers separately:

```
columns = np.array([
    "EmployeeID", "Name", "Department", "Salary", "JoinDate", "City", "Bonus"
])

print(columns)
```

4️⃣ Access Data (Examples)

Access a row

```
print(data[0])
```

Access a column (Salary)

```
salary_column = data[:, 3]
print(salary_column)
```

Find fully empty rows

```
empty_rows = [row for row in data if all(x is None for x in row)]
print(empty_rows)
```

5️⃣ Save NumPy Table to CSV (Optional but useful)

```
np.savetxt(
    "employees_numpy.csv",
    data,
    delimiter=",",
    fmt="%s"
)
```

📌 Headers won't be included automatically.
(That's why Pandas is preferred in real projects.)

6️⃣ Very Important Interview Point 🧠

❓ Why NumPy is not ideal for this?

- ❌ No column names
- ❌ Poor handling of mixed data
- ❌ Harder null handling

✅ Where NumPy is used?

- Numerical computation
- Matrices
- ML preprocessing
- Scientific calculations

> 👉 **Best practice:**
> **NumPy → Pandas → SQL / Power BI**

🔥 One-Line Interview Answer

> “I used NumPy to store raw data arrays, then converted them to Pandas DataFrames for structured analysis and cleaning.”

If you want next:

- ➡️ Convert **NumPy → Pandas**
- ➡️ Clean NULLs using NumPy
- ➡️ Power BI import from NumPy CSV
- ➡️ Project-ready explanation for resume

Just say **next** 🚀

Broadcasting

BroadCasting

1) Why Broadcasting imp—

When we add or subtract 2 array which have different shape… then at that time we could unable to do that… that is knowns by Broadcasting like we can perform these things on only specific arrays which follows rules…

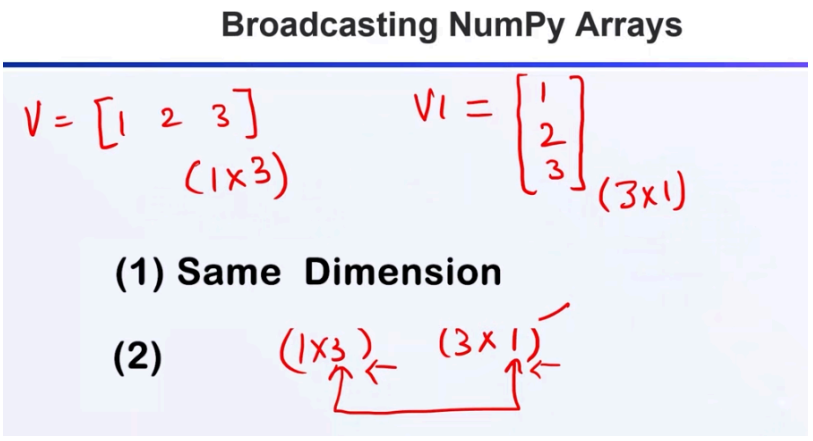
Eg -

```
x = np.array([23,4,3,2])
y = np.array([2,3,5])

print(x+y)
```

0.1s

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[11], line 4
      1 x = np.array([23,4,3,2])
      2 y = np.array([2,3,5])
----> 4 print(x+y)

ValueError: operands could not be broadcast together with shapes (4,) (3,)
```

2) Rule



Basically 2 rules for 1D (if any one is 1D) -

1) **Dimensions should be same…**
2) **or, one of them columns (r*c) should be 1.**

And 2 rules for 2D (if both is 2D) – not sure…

3) **one of them columns (r*c) should be 1.**
4) **And another element (row) should be of same size (r*c = r*c)**

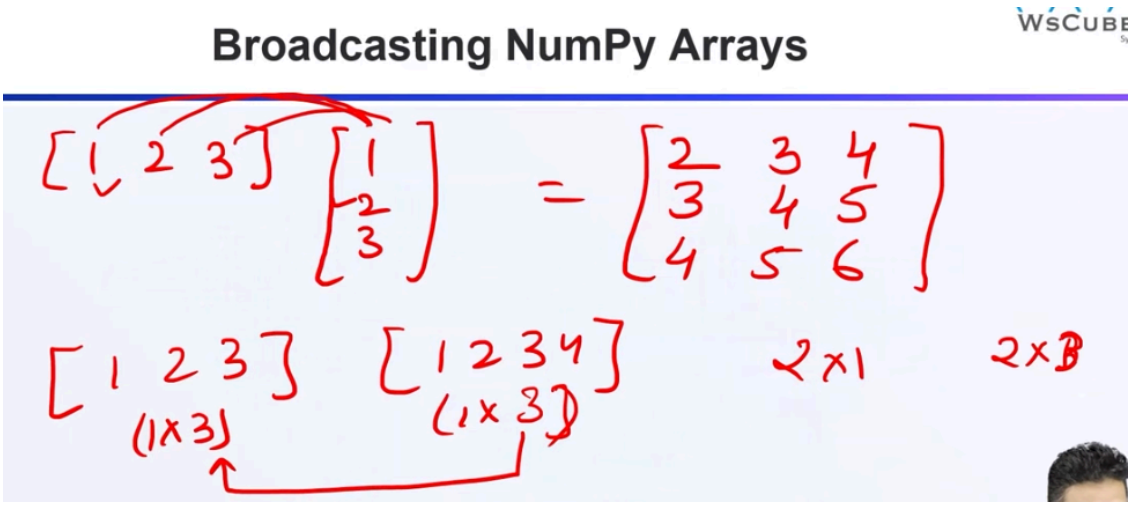
3) Eg 1 -

```
var1 = np.array([1,2])
```

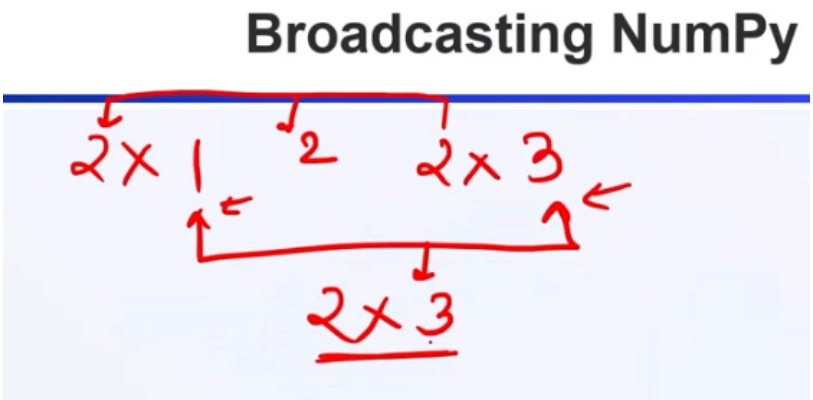
```
print(var1.shape,"\n")
print(var1)
print(var1.ndim,"\n")

var2 = np.array([[1],[2],[3]])
print(var2.shape,"\n")
print(var2)
print(var2.ndim,"\n")

print(var1+var2)
```

```
Output -
(2,)

[1 2]
1

(3, 1)

[[1]
 [2]
 [3]]
2

[[2 3]
 [3 4]
 [4 5]]
```

Sol 1-



And how to decide final or product shape-



R X C → (max no of R) X (max no of C)

Eg 2- (may be)
And 2 rules for 2D (if both is 2D)

5) **one of them columns (r*c) should be 1.**
6) **And another element (row) should be of same size (r*c = r*c)**

```
x = np.array([[1],[2],[3],[4],[5],[6]])
print(x.shape)

y =
np.array([[1,2,3],[1,2,3],[1,2,3],[1,2,3],[1,2,3],[1,2,3]])
print(y.shape)

print(x+y)
```

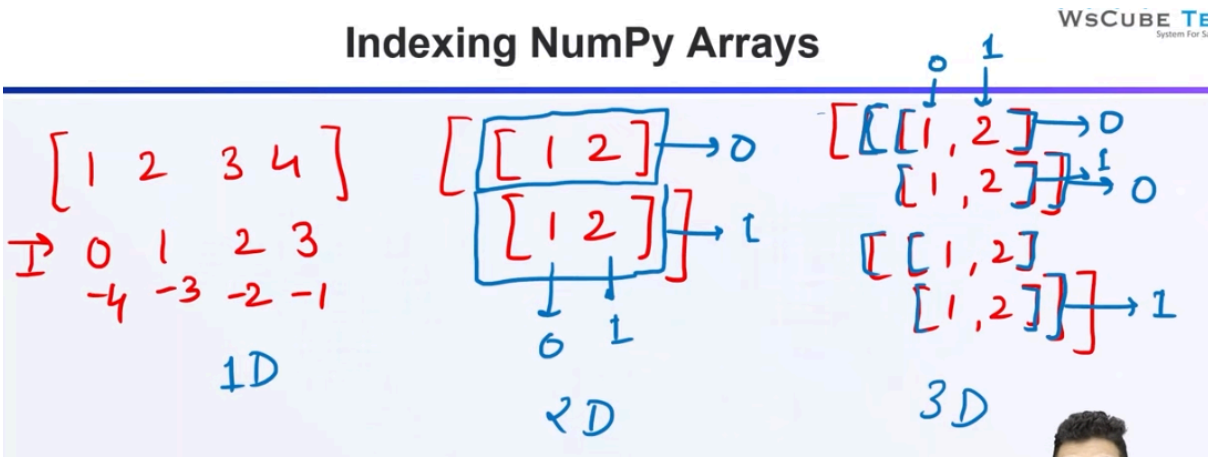
Output -

```
(6, 1)
(6, 3)
[[2 3 4]
 [3 4 5]
 [4 5 6]
 [5 6 7]
 [6 7 8]
 [7 8 9]]
```

4) **Daslfj**

Indexing & Slicing

Indexing And Slicing



1) Indexing -

Eg -

```
#for 1D
var = np.array([1,2,4])
#                0,1,2
#               -3,-2,-1

print(var,"\n")
print(var[2])
print(var[-1])
```

```
Output -
[1 2 4]

4
4
```

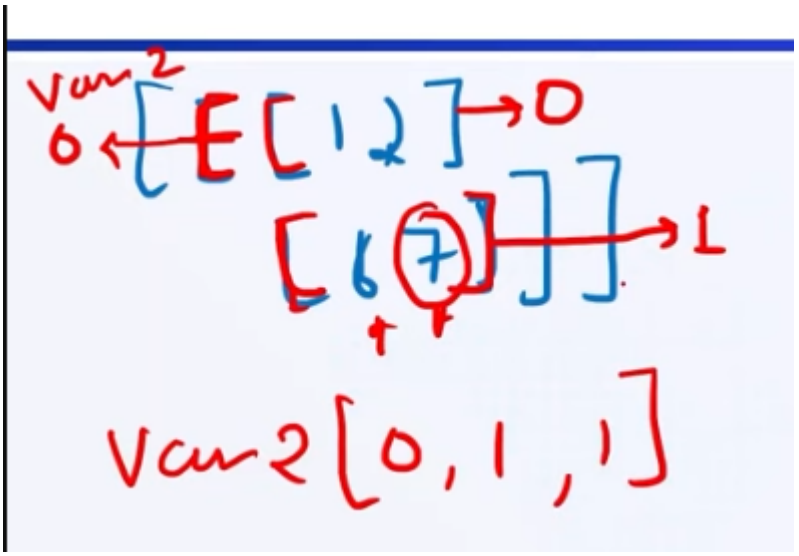
```
#for 2D
var1 = np.array([[1,2,3],[4,5,6]])

print(var1)
print(var1.ndim,"\n")
print(var1[1,2]) #-- for 6
```

```
Output -
[[1 2 3]
 [4 5 6]]
2
```

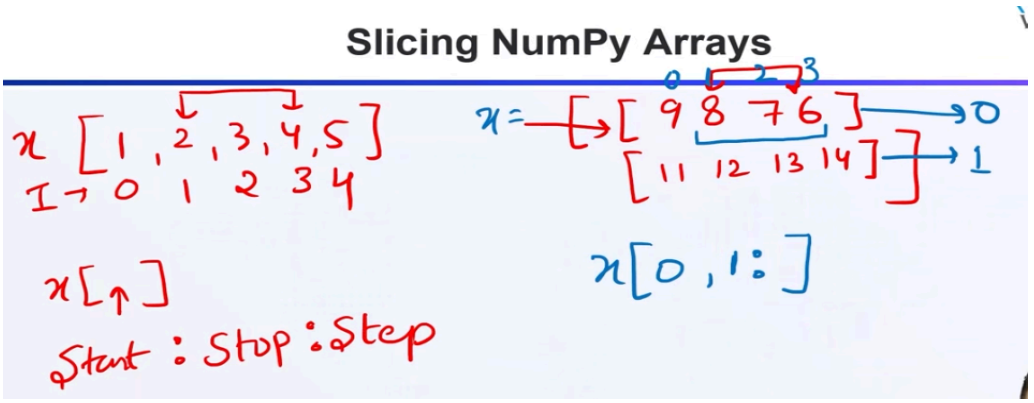
```
6
```

```
#for 3D
var2 = np.array([[[1,2,3],[4,5,6]]])

print(var2)
print("Dimension :",var2.ndim,"and Shape :",var2.shape,"\n")
print(var2[0,1,2]) #-- for 6

Output -
[[[1 2 3]
  [4 5 6]]]
Dimension : 3 and Shape : (1, 2, 3)

6
```

_given below explanation for 3d eg.



2) Slicing -



Slicing means - extract some part by reference through index number, formula - var[start index : last index +1 : steps]

Eg -

1D

```
var = np.array([9,0,7,1,6,2,4,3,5])
# indexing --   0,1,2,3,4,5,6,7,8

print(var)
print("shape :",var.shape,"dim :",var.ndim,"\n")
print("extract 0 to 2 :",var[1:6])
print("7 to end :",var[2:])
print("start to 3",var[:-1])
print("Steps :",var[::4])
print("steps b/w 7 and 3 stps of 3 :",var[2:8:3])
```

Output -

```
[9 0 7 1 6 2 4 3 5]
shape : (9,) dim : 1

extract 0 to 2 : [0 7 1 6 2]
7 to end : [7 1 6 2 4 3 5]
start to 3 [9 0 7 1 6 2 4 3]
Steps : [9 6 5]
steps b/w 7 and 3 steps of 3 : [7 2]
```

2D

```
var1 = np.array([[1,2,3,7,8],[4,5,6,0,9]])

print(var1)
print("shape :",var1.shape,"dim :",var1.ndim,"\n")
print("extract 5 to 0:",var1[1,1:6]) # or var1[1,1:-1]
print("Extract 2 to end with steps 3",var1[0,1::3])
```

Output -

```
[[1 2 3 7 8]
 [4 5 6 0 9]]
shape : (2, 5) dim : 2

extract 5 to 0: [5 6 0 9]
Extract 2 to end with steps 3 [2 8]
```

3D

```
var2 = np.array([[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]])
var3 = var2.reshape(2,2,3)
print(var3)
print("shape :",var3.shape,"dim :",var3.ndim,"\n")

print("extract 10 to end",var3[1,1,0:])
print("extract 7 to end, by step 2",var3[1,0,0::2])
```

```
Output -
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
shape : (2, 2, 3) dim : 3

extract 10 to end [10 11 12]
extract 7 to end, by step 2 [7 9]
```

3) If you want to 3d shape like - (`(2, 2, 3)`) without reshape, so

```
var3on = np.array([
    [[1,2,3],[4,5,6]],
    [[7,8,9],[10,11,12]]
])

print(var3on)
print("shape :",var3on.shape,"dim :",var3on.ndim,"\n")
```

Output -

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
shape : (2, 2, 3) dim : 3
```

4) adf

Iterating, copy_view

ITERATING | COPY and VIEW

1) ITERATING

a) Simple way - Without any specific method.

```
var = np.array([9,8,7,6,5,4,3])
print(var,"\n")
for i in var:
    print(i)
```

```
[9 8 7 6 5 4 3]

9
8
7
6
5
4
3
```

```
var1 = np.array([[1,2,3,4],[5,6,7,8]])
print(var1,"\n")

#two ways --
# i) it will show like dimensions wise
for i in var1:
    print(i,"\n")

# ii) Now this is best way for items...
for i in var1:
    for k in i:
        print(k)
```

```
[[1 2 3 4]
 [5 6 7 8]]

[1 2 3 4]

[5 6 7 8]

1
2
3
4
```

```
5
6
7
8

var2 =
np.array([[[1,2,3,4],[5,6,7,8]],[[9,10,11,12],[13,14,1
5,16]]])
print(var2,"\n")
print("Shape :",var2.shape,"and dim :",var2.ndim,"\n")

for i in var2:
    print(i,"\n")

for i in var2:
    for k in i:
        for l in k:
            print(l)

[[[ 1  2  3  4]
  [ 5  6  7  8]]

 [[ 9 10 11 12]
  [13 14 15 16]]]

Shape : (2, 2, 4) and dim : 3

[[1 2 3 4]
 [5 6 7 8]]

[[ 9 10 11 12]
 [13 14 15 16]]

1
2
3
4
5
6
7
8
9
10
11
```

```
...
13
14
15
16
```

b) With Methods… (nditer() and **ndenumerate())**

Nditer = for iterator

Ndenumerate = for index number and iterator

Eg 1-

```
var1 = np.array([[1,2,3,7,8],[4,5,6,0,9]])
print(var1,"\n")
print("Shape :",var1.shape,"Dim :",var1.ndim,"\n")

for i in np.nditer(var1):
    print(i)
```

Output -

```
[[1 2 3 7 8]
 [4 5 6 0 9]]

Shape : (2, 5) Dim : 2
1
2
3
7
8
4
5
6
0
9
```

Eg 2-

```
var2 =
np.array([[[1,2,3,4],[5,6,7,8]],[[9,10,11,12],[13,14,15,
16]]])
for i in np.nditer(var2):
    print(i)
```

Output - full element…

Eg 3 - (**Nditer)** if you want to change data type also…

`flags=['buffered'],op_dtypes=["S"]`

```
var1 = np.array([[1,2,3,7,8],[4,5,6,0,9]])
print(var1,"\n")
print("Shape :",var1.shape,"Dim :",var1.ndim,"\n")

for i in
np.nditer(var1,flags=['buffered'],op_dtypes=["S"]):
    print(i)

Output -
[[1 2 3 7 8]
 [4 5 6 0 9]]

Shape : (2, 5) Dim : 2

b'1'
b'2'
b'3'
b'7'
b'8'
b'4'
b'5'
b'6'
b'0'
B'9'
```

✅ Meaning of each parameter

🔹 `flags=['buffered']`

- **Purpose:** Allow temporary buffered storage while iterating
- **Why needed:** Because the loop wants to **change dtype** from original (int) to another type (string "S").
- NumPy cannot directly convert while iterating, so it uses a temporary buffer.

📌 Simple meaning:
"Temporary space banao taaki data convert hokar safely iterate ho sake."

🔹 `op_dtypes=["S"]`

- **op_dtypes = operand data types**
- `"S"` → **String dtype** (bytes string, not python str)
- Forces every element to be read as a **string** instead of int.

📌 Simple meaning:
"Aap array ke numbers ko string jaisa treat karna chahte ho."

✅ Short Summary

Parameter	Meaning	Explanation
`buffered`	Use temporary buffer during iteration	"Convert karte time crash na ho, ek buffer bana do."
`op_dtypes=["S"]`	Convert each element to string while iterating	"Integers ko string ke format me iterate karo."

🧠 Final One-liner (Hinlgish)

buffered = safe conversion ke liye temporary storage
op_dtypes = iterate karte waqt data ko kis type me treat karna hai

Eg 4 - **(Ndenumerate)**

```
var3 = np.array([[[1,2,3,4],[5,6,7,8]]])
print("shape :",var3.shape,"dim :",var3.ndim,"\n")

for i,d in np.ndenumerate(var3):
    print(i,d)
```

Output -

```
shape : (1, 2, 4) dim : 3

((0, 0, 0), 1)
((0, 0, 1), 2)
((0, 0, 2), 3)
((0, 0, 3), 4)
((0, 1, 0), 5)
((0, 1, 1), 6)
((0, 1, 2), 7)
```

```
((0, 1, 3), 8)
```

2) Copy Vs Views

a) Copy

```
var = np.array([1,2,3,4])
co = var.copy()  #or -- np.copy(var)

var[2] = 21

print("var :",var)
print("copy :",co)
```

Output -

```
var : [ 1  2 21  4]
copy : [1 2 3 4]
```

b) Views

```
x = np.array([1,2,3,4])
vi = x.view() #--only this way

x[2] = 21

print("x :",x)
print("view :",vi)
```

Output -

```
x : [ 1  2 21  4]
view : [ 1  2 21  4]
```

c) Difference

Copy vs View in NumPy Arrays

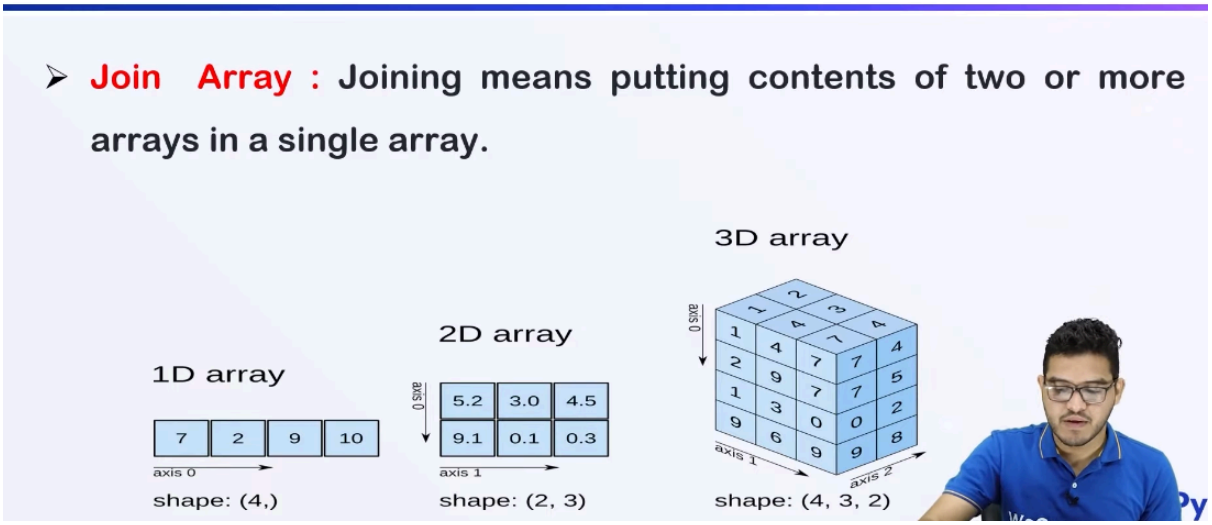
The Difference Between Copy and View :

The copy owns the data.	The view does not own the data .
The copy of an array is a new array.	A view of the original array.
The changes made in the copy data does not reflect in the original array.	any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

Join & Split

Join and Split

1) Join



Eg 1 (1D)

```
var = np.array([1,2,3,4])
var1 = np.array([5,6,7,8]) #var1 = np.array([5,6,7])
#Not need to enter same no of element in both array

ar = np.concatenate((var,var1))
print(ar)
```

```
Output -
[1 2 3 4 5 6 7 8]
```

Eg 2 (2D)

```
vr = np.array([[1,2],[3,4]])
vr1 = np.array([[5,9],[7,8]])

ar_new = np.concatenate((vr,vr1))
#but in 2D same no of element required
ar_axis = np.concatenate((vr,vr1),axis=0)
ar_axi = np.concatenate((vr,vr1),axis=1)
print(ar_new,"\n")
print(ar_axis,"\n")
print(ar_axi,"\n")
```

```
Output -
[[1 2]
 [3 4]
```

```
 [5 9]
 [7 8]]

[[1 2]
 [3 4]
 [5 9]
 [7 8]]

[[1 2 5 9]
 [3 4 7 8]]
```

For AXIS there is also another Method Called STACK, in which we don't required to enter axis = 0 or 1,

Eg - 1

```
var_1 = np.array([1,2,3,4])
var_2 = np.array([5,6,7,8])

a_new = np.stack((var_1,var_2),axis = 0)
print(a_new,"\n")
a_new1 = np.hstack((var_1,var_2)) #row
print(a_new1,"\n")
a_new2 = np.vstack((var_1,var_2)) #columns
print(a_new2,"\n")
a_new3 = np.dstack((var_1,var_2)) #height
print(a_new3,"\n")
print(var_1,var_2)
```

```
Output -
[[1 2 3 4]
 [5 6 7 8]]

[1 2 3 4 5 6 7 8]

[[1 2 3 4]
 [5 6 7 8]]

[[[1 5]
  [2 6]
  [3 7]
```

```
  [4 8]]]

[1 2 3 4] [5 6 7 8]
```

Output -

2) Split

Splitting breaks one array into multiple

Eg 1 -

```
var = np.array([1,2,3,4,5,6])

print(var)
ar = np.split((var),2)
print(ar)
print(type(ar))
print(ar[1])
```

```
Output -
[1 2 3 4 5 6]
[array([1, 2, 3]), array([4, 5, 6])]
<class 'list'>
[4 5 6]
```

Eg 2 -

```
vr_new = np.array([[1,2],[3,4],[5,6]])

print(vr_new,"\n")
ar1 = np.array_split(vr_new,3)
ar2 = np.array_split(vr_new,2,axis=1)
print(ar1,"\n")
print(ar2)
```

Output -

```
[[1 2]
 [3 4]
 [5 6]]

[array([[1, 2]]), array([[3, 4]]), array([[5, 6]])]
```

```
[array([[1],
        [3],
        [5]]), array([[2],
        [4],
        [6]])]
```

And same eg if you enter split more than sufficient -

```
vr_new = np.array([[1,2],[3,4],[5,6]])

print(vr_new,"\n")
ar1 = np.array_split(vr_new,3)
ar2 = np.array_split(vr_new,3,axis=1)
print(ar1,"\n")
print(ar2)
```

✓ 0.0s

```
[[1 2]
 [3 4]
 [5 6]]

[array([[1, 2]]), array([[3, 4]]), array([[5, 6]])]

[array([[1],
       [3],
       [5]]), array([[2],
       [4],
       [6]]), array([], shape=(3, 0), dtype=int32)]
```

Difference b/w 3 and 4…

```
vr_new = np.array([[1,2],[3,4],[5,6]])

print(vr_new,"\n")
ar1 = np.array_split(vr_new,3)
ar2 = np.array_split(vr_new,4,axis=1)
print(ar1,"\n")
print(ar2)
```

✓ 0.0s

```
[[1 2]
 [3 4]
 [5 6]]

[array([[1, 2]]), array([[3, 4]]), array([[5, 6]])]

[array([[1],
       [3],
       [5]]), array([[2],
       [4],
       [6]]), array([], shape=(3, 0), dtype=int32), array([], shape=(3, 0), dtype=int32)]
```

3) daf

srch,srt,fltr

Search | Sort | Search Sorted | Filter

1) Search

- **Search Array :** Search an array for a certain value, and return the indexes that get a match.

Eg - 1

```
var = np.array([9,23,2,4,2,8,1,1,2,3,2,4])
sr = np.where(var == 2) #--it will give index no of 2
print(sr)
```

Output -

```
(array([ 2,  4,  8, 10], dtype=int64),)
```

Eg - 2

```
var = np.array([9,23,2,4,2,8,1,1,2,3,2,4])
sr = np.where((var%2) == 0) #--it will give index no of 2
print(sr)
```

Output -

```
(array([ 2,  3,  4,  5,  8, 10, 11], dtype=int64),)
```

```
For 2 D
var = np.array([[9,23,2,4,2,8],[1,1,2,3,2,4]])

sr = np.where(var == 2) #--it will give index no of 2
print(sr)

(array([0, 0, 1, 1], dtype=int64), array([2, 4, 2, 4],
dtype=int64))
```

2) Search Sorted

- **Search Sorted Array :** which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

Eg - 1

```
var1 = np.array([1,2,4,5,6,7,9])
#index            0,1,2,3,4,5,6

ar = np.searchsorted(var1,3) #tell best index no for fit it
print(ar)
```

```
2
```

Eg- a) – Also use as list

```
var1 = np.array([1,2,4,5,6,7,9])
#index            0,1,2,3,4,5,6

ar = np.searchsorted(var1,[7,1,4]) #tell best index no for
fit it
print(ar)
```

```
[5 0 2]
```

Eg - b) — Insert elements in original array….

```
var1 = np.array([1,2,4,5,6,7,9])
#index            0,1,2,3,4,5,6

ar = np.searchsorted(var1,[7,1,4]) #tell best index no for
fit it
print(ar)

#and if i want to Insert values into array at correct
positions so,
var_update = np.insert(var1,ar,[7,1,4])
print(var_update)
```

```
Output -
[5 0 2]
[1 1 2 4 4 5 6 7 7 9]
```

```
Eg - c), (left - Right)
var2 = np.array([1,2,4,5,6,7,9])
#index            0,1,2,3,4,5,6
```

```
x1 = np.searchsorted(var2,2, side= 'right')
print(x1)

Output -
2… for left - 1
```

Value exists in array?	side='left'	side='right'
No (like 3)	insert before first greater value	insert before first greater value
Yes (like 7)	before equal value	after equal value

3) Sort

➢ **Sort Array :** Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

Eg - 1

```
var = np.array([9,23,2,4,2,8,1,1,2,3,2,4])
print(np.sort(var))
```

Output -

```
[ 1  1  2  2  2  2  3  4  4  8  9 23]
```

Eg - 2

```
var_2d = np.array([[9,23,2,4],[2,8,1,1],[2,3,2,4]])
print(np.sort(var_2d))
```

Output -

```
[[ 2  4  9 23]
 [ 1  1  2  8]
 [ 2  2  3  4]]
```

Eg - 3

```
var_cap = np.array(['s','a','i','z','y'])
print(np.sort(var_cap))
```

Output -

```
['a' 'i' 's' 'y' 'z']
```

4) Filter

- **Filter Array :** Getting some elements out of an existing array and creating a new array out of them.

Eg - 1

```
var_fil = np.array(['s','a','i','z','y'])
fil = [True,True,False,True,False]

new_a = var_fil[fil]
print(new_a)
```

Output -

```
['s' 'a' 'z']
```

5) daf

sfl,unq,rsz,flt,rvl_fn

Numpy Array Functions…

Shuffle | Unique | Resize | Flatten | Ravel Functions

1) Shuffle

```
var = np.array([1,2,3,4,5])

np.random.shuffle(var)
print(var)
```

```
Output -
[3 2 5 4 1]
```

2) Unique

```
var1 = np.array([1,2,4,4,1,5,6,102,102,4,5])

ar = np.unique(var1)
print(ar)
```

```
Output -
[  1   2   4   5   6 102]
```

1) If also want Index number along with it

```
var1 = np.array([1,2,4,4,1,5,6,102,102,4,5])

ar = np.unique(var1,return_index=True)
print(ar)
```

Output -

```
(array([  1,   2,   4,   5,   6, 102]), array([0, 1, 2, 5,
6, 7], dtype=int64))
```

2) And if you want also, count (no of elements outcoming)

```
var1 = np.array([1,2,4,4,1,5,6,102,102,4,5])

ar = np.unique(var1,return_index=True,return_counts=True)
print(ar)
```

Output -

```
(array([  1,   2,   4,   5,   6, 102]), array([0, 1, 2, 5,
6, 7], dtype=int64), array([2, 1, 3, 2, 1, 2], dtype=int64))
```

3) Resize - (conv size 1D, 2D, 3D accor…)

```
var2 = np.array([1,2,3,4,5,6])
print(ar_rz.ndim)
ar_rz = np.resize(var2,(3,2))
print(ar_rz)

Output -
2
[[1 2]
 [3 4]
 [5 6]]  and for - (2,3) so - row 2 and column - 3
```

What if??? we enter greater size than total element

In short - Nothing …(it will use again same element)

```
var2 = np.array([1,2,3,4,5])

ar_rz = np.resize(var2,(3,3))
print(ar_rz)

Output -
[[1 2 3]
 [4 5 1]
 [2 3 4]]
```

Eg 2– make 2d or 3 d

```
var2 = np.array([1,2,3,4,5,6])

ar_rz = np.resize(var2,(2,2,2))
print(ar_rz.ndim)
print(ar_rz)

Output -
3

[[[1 2]
  [3 4]]

 [[5 6]
  [1 2]]]
```

4) Flatten - conv 2D or 3D in 1D acc Size

Order : {'C', 'F', 'A', 'K'}, Optional

- 'C' means to flatten in row-major (C-style) order.
- 'F' means to flatten in column-major (Fortran-style) order.
- 'A' means to flatten in column-major order if `a` is Fortran *contiguous* in memory,row-major order otherwise.
- 'K' means to flatten `a` in the order the elements occur in memory.
- The default is 'C'.

Eg 1

```
print(ar_rz,"\n")
print(ar_rz.flatten())
```

```
[[[1 2]
  [3 4]]

 [[5 6]
  [1 2]]]

[1 2 3 4 5 6 1 2]
```

Eg 1 (when order = “c”)

```
print(ar_rz,"\n")
y = np.array([[1,2],[3,4]])
print(ar_rz.flatten(order="c"),"\n")
print(y,"and dim :",y.ndim,"\n")
print(y.flatten(order="c"))
```

25] ✓ 0.0s

```
[[[1 2]
  [3 4]]

 [[5 6]
  [1 2]]]

[1 2 3 4 5 6 1 2]

[[1 2]
 [3 4]] and dim : 2

[1 2 3 4]
```

Eg 2 (when order = “f”)

```
print(ar_rz,"\n")
y = np.array([[1,2],[3,4]])
print(ar_rz.flatten(order="f"),"\n")
print(y,"and dim :",y.ndim,"\n")
print(y.flatten(order="f"))
```

[26] ✓ 0.0s

```
[[[1 2]
  [3 4]]

 [[5 6]
  [1 2]]]

[1 5 3 1 2 6 4 2]

[[1 2]
 [3 4]] and dim : 2

[1 3 2 4]
```

5) Ravel - (conv 2D or 3D in 1D acc Size)

```
print(ar_rz,"\n")
print(ar_rz.ravel())
Output -
[[[1 2]
  [3 4]]
 [[5 6]
  [1 2]]]
[1 2 3 4 5 6 1 2]
```

Eg 1 (when order = “c”)

```
print(ar_rz,"\n")
y = np.array([[1,2],[3,4]])
print(ar_rz.ravel(order="C"),"\n")
print(y,"and dim :",y.ndim,"\n")
print(y.ravel(order="C"))
```

[31] ✓ 0.0s

```
[[[1 2]
  [3 4]]

 [[5 6]
  [1 2]]]

[1 2 3 4 5 6 1 2]

[[1 2]
 [3 4]] and dim : 2

[1 2 3 4]
```

Eg 2 (when order = “f”)

```
    print(ar_rz,"\n")
    y = np.array([[1,2],[3,4]])
    print(ar_rz.ravel(order="f"),"\n")
    print(y,"and dim :",y.ndim,"\n")
    print(y.ravel(order="f"))
```

32] ✓ 0.0s

```
[[[1 2]
  [3 4]]

 [[5 6]
  [1 2]]]

[1 5 3 1 2 6 4 2]

[[1 2]
 [3 4]] and dim : 2

[1 3 2 4]
```

Feature	ravel()	flatten()
Returns	View (if possible)	Always a new copy
Memory usage	Efficient	Uses extra memory
Changes affect original array?	✅ Yes (if view)	❌ No
Speed	Faster	Slightly slower
Parameter options	order='C'/'F'	Same

Use ravel() when…	Use flatten() when…
You want a fast view	You need a true copy
Memory efficiency matters	You don’t want original to change
You are okay with modification affecting original	You want data independence

➜ **ravel()** → view (no copy if possible)
➜ **flatten()** → copy (always)

6) adf

Insert, Delete

Insert | Delete

1) Insert

Eg 1

```
var = np.array([1,2,3,4])
a = np.insert(var,0,30)
print(var)
print(a)
```

```
Output -
[1 2 3 4]
[30  1  2  3  4]
```

Eg 2

```
var = np.array([1,2,3,4])
a = np.insert(var,[2,3],(30,2)) # can be use (2,3) or [2,3]
print(a)
```

```
[ 1  2 30  3  2  4]
```

Eg 3 (always take integer, not take bool)

```
var = np.array([1,2,3,4])
a = np.insert(var,0,5.98)
print(a)
```

```
[5 1 2 3 4]
```

Eg 4 (2_D) and axis = 0 (for row) , axis = 1 (for columns)

```
new_y = np.array([[1,2],[3,4]])
new_a = np.insert(new_y,0,(46,34))
print(new_a)
```

[49] ✓ 0.0s

```
[46 34  1  2  3  4]
```

```
new_y = np.array([[1,2],[3,4]])
new_a = np.insert(new_y,0,(46,34),axis=1)
print(new_a)
```

[51] ✓ 0.0s

```
[[46  1  2]
 [34  3  4]]
```

```
new_y = np.array([[1,2],[3,4]])
new_a = np.insert(new_y,0,(46,34),axis=0)
print(new_a)
```

[52] ✓ 0.0s

```
[[46 34]
 [ 1  2]
 [ 3  4]]
```

2) Append

Work same as Insert, but it only insert element at the end…

Eg1 -

```
var = np.array([1,2,3,4])
app = np.append(var,5.98)
print(app)
```

```
Output -
[1.   2.   3.   4.   5.98]
```

Eg2 -

```
new_y = np.array([[1,2],[3,4]])
#print(new_y,new_y.ndim,new_y.shape)

x = np.append(new_y,[[5,6]],axis=0)
y = np.append(new_y,[[5],[6]],axis=1)
print(x,"\n")
print(y)
```

[100] ✓ 0.0s

```
[[1 2]
 [3 4]
 [5 6]]

[[1 2 5]
 [3 4 6]]
```

```
new_y = np.array([[1,2,6],[3,4,5]])
z = np.append(new_y,[[46],[22]], axis=1)
a = np.append(new_y,[[46,32,22]], axis=0)
print(z,"\n")
print(a)
```

[102] ✓ 0.0s

```
[[ 1  2  6 46]
 [ 3  4  5 22]]

[[ 1  2  6]
 [ 3  4  5]
 [46 32 22]]
```

3) Delete

```
var = np.array([1,2,3,4])

vd = np.delete(var,0) #enter index no
print(var)
print(vd)

Output
[1 2 3 4]
[2 3 4]

Eg
var_2d = np.array([[1,2],[3,4]])


vd_2d = np.delete(var_2d,3)
print(vd_2d,"dim :",vd_2d.ndim)

vd_2d = np.delete(var_2d,(0,2))
print(vd_2d)
#it delete index numbers…

Output -
[1 2 3] dim : 1
[2 4]
```

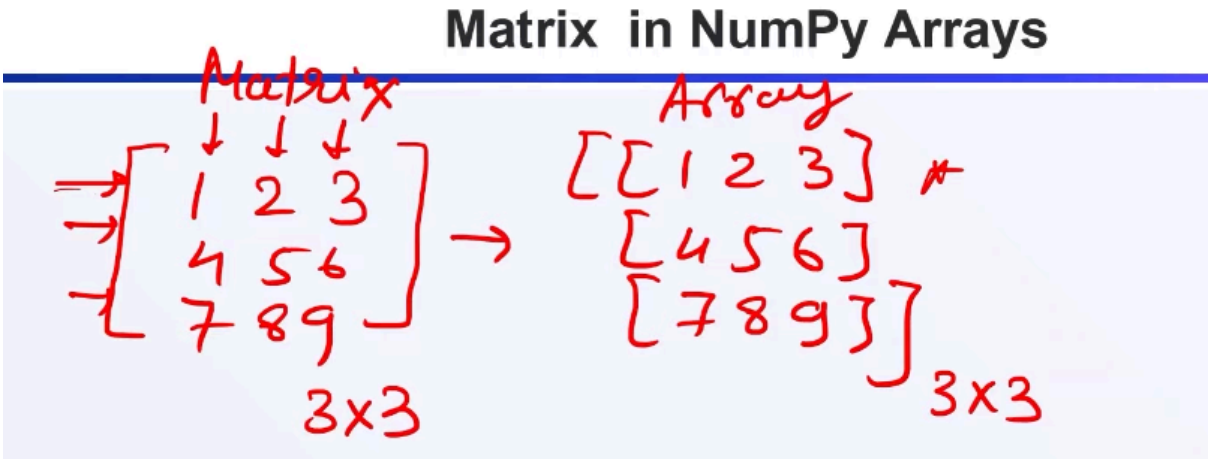
4) Ljdaf

5) df

Matrix & it's f'n

Matrix

1) Difference b/w Matrix



There is many thing same and many difference like
Same - add, subs
Different - Multiplications

```
var = np.matrix([[1,2,3],[4,5,6]])

print(var)
print(type(var))
print(var+var)
print(var-var)
```

[106] ✓ 0.0s

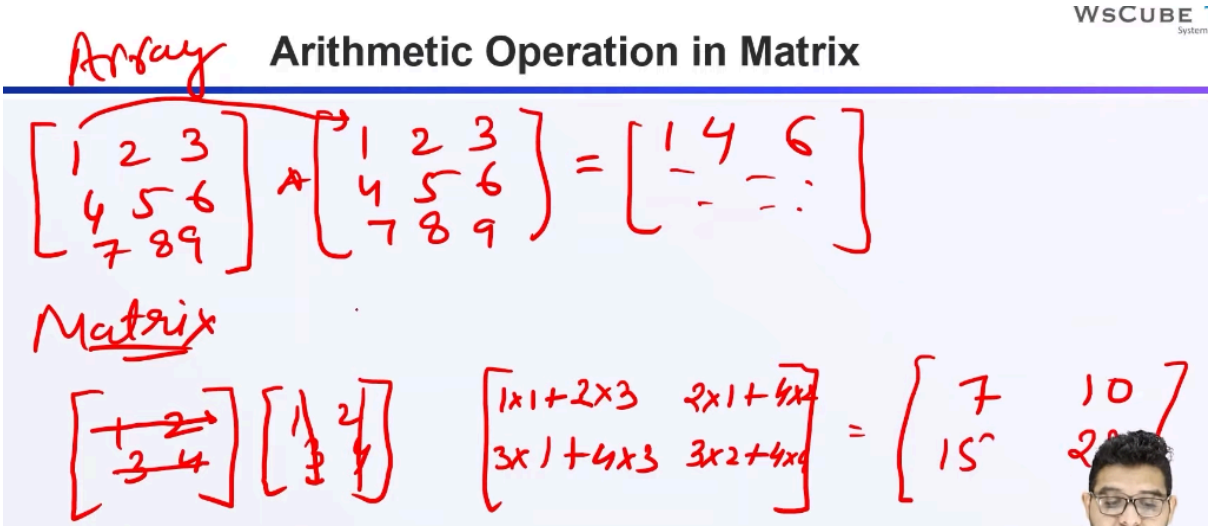
```
[[1 2 3]
 [4 5 6]]
<class 'numpy.matrix'>
[[ 2  4  6]
 [ 8 10 12]]
[[0 0 0]
 [0 0 0]]
```

```
var_ar = np.array([[1,2,3],[4,5,6]])

print(var_ar)
print(type(var_ar))
print(var_ar+var_ar)
print(var_ar-var_ar)
```

[105] ✓ 0.0s

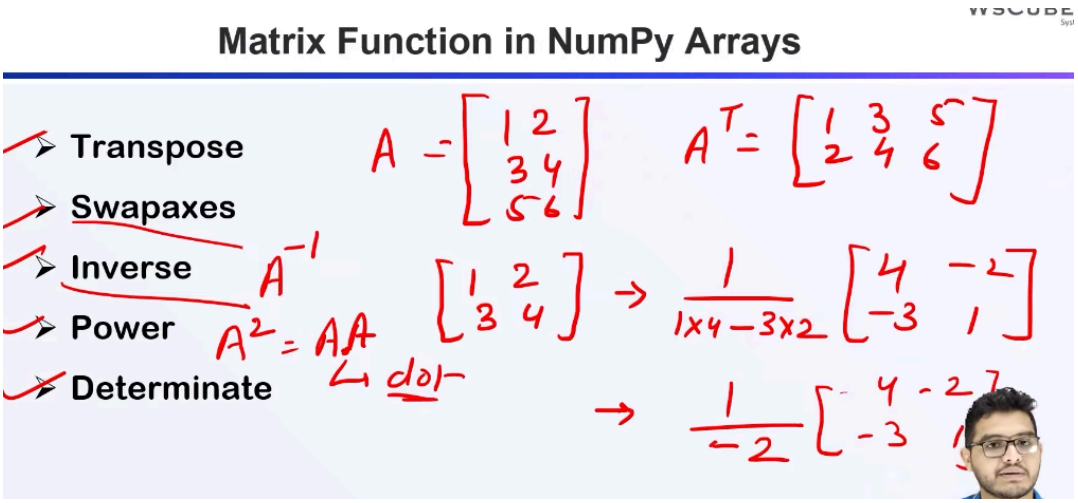
```
[[1 2 3]
 [4 5 6]]
<class 'numpy.ndarray'>
[[ 2  4  6]
 [ 8 10 12]]
[[0 0 0]
 [0 0 0]]
```

For Multiplication




```
var1 = np.matrix([[1,2],[1,2]])
var2 = np.matrix([[1,2],[1,2]])
print(var1,"\n") #--- same as var2
print(var1*var2,"\n")
print(var1.dot(var2))
Output -
[[1 2]
 [1 2]]

[[3 6]
 [3 6]]

[[3 6]
 [3 6]]
```

2) Matrix Functions



1) Transpose

Eg - 1

```
tr_vr = np.matrix([[1,2,3],[4,5,6]])
tr_a = np.transpose(tr_vr)
print(tr_vr,"\n")
print(tr_a)
```

```
Output-
[[1 2 3]
 [4 5 6]]

[[1 4]
 [2 5]
 [3 6]]
```

Eg - 2 - also have short method...

```
tr_vr1 = np.matrix([[1,2,3],[4,5,6]])

print(tr_vr1,"\n")
print(np.transpose(tr_vr1),"\n") #or
print(tr_vr1.T)
```

Output -

```
[[1 2 3]
 [4 5 6]]

[[1 4]
 [2 5]
 [3 6]]

[[1 4]
 [2 5]
 [3 6]]
```

2) Swapaxes (It's work same as Transpose)

Eg - 1

```
tr_vr1 = np.matrix([[1,2,3],[4,5,6]])

print(tr_vr1,"\n")
print(np.swapaxes(tr_vr1,0,1),"\n")
print(np.swapaxes(tr_vr1,1,1))
```

```
#-- (1,1) and (0,0) work same ,and - (1,0) or (0,1) work same
```

```
Output -
[[1 2 3]
 [4 5 6]]

[[1 4]
 [2 5]
 [3 6]]

[[1 2 3]
 [4 5 6]]
```

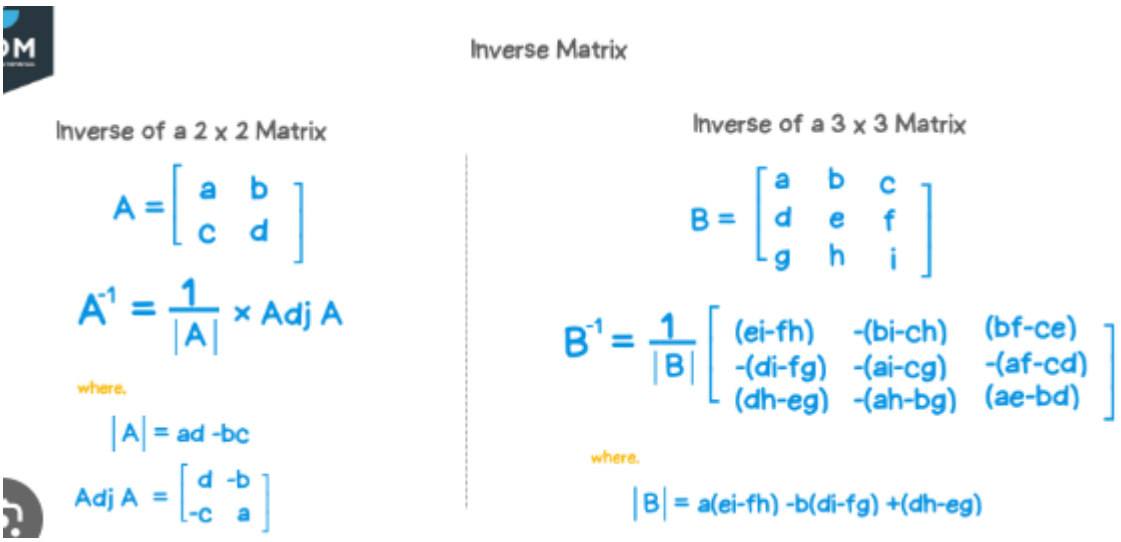
Eg - 2

```
sp_vr1 = np.matrix([[1,2],[3,4]])
print(sp_vr1,"\n")
print(np.swapaxes(sp_vr1,0,1))
```

Output -

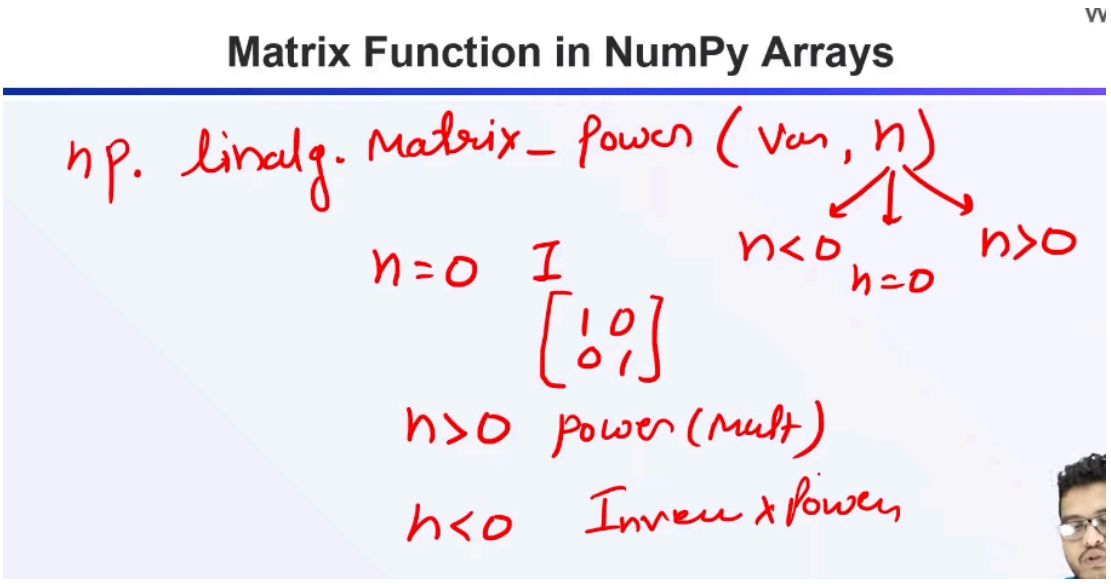
```
[[1 2]
 [3 4]]

[[1 3]
 [2 4]]
```

3) Inverse (np.linalg.inv(inv_vr))


Inverse Matrix
Inverse of a 2 x 2 Matrix
A = [a b; c d]
A^-1 = 1/|A| × Adj A
where,
|A| = ad -bc
Adj A = [d -b; -c a]
Inverse of a 3 x 3 Matrix
B = [a b c; d e f; g h i]
B^-1 = 1/|B| [(ei-fh) -(bi-ch) (bf-ce); -(di-fg) (ai-cg) -(af-cd); (dh-eg) -(ah-bg) (ae-bd)]
where,
|B| = a(ei-fh) -b(di-fg) +(dh-eg)

```
inv_vr = np.matrix([[1,2],[3,4]])
print(inv_vr,"\n")
print(np.linalg.inv(inv_vr))

Output -
[[1 2]
 [3 4]]

[[-2.   1. ]
 [ 1.5 -0.5]]
```

4) Power




```
pw_vr = np.matrix([[1,2],[3,4]])

print(pw_vr,"\n")
print(np.linalg.matrix_power(pw_vr,0),"\n")
print(np.linalg.matrix_power(pw_vr,2),"\n") #--Pow
Post
print(np.linalg.matrix_power(pw_vr,-2)) #--Pow Neg

Output-
[[1 2]
 [3 4]]

[[1 0]
```

```
 [0 1]]

[[ 7 10]
 [15 22]]

[[ 5.5  -2.5 ]
 [-3.75  1.75]]
```

For Power in negative -

A^{-n} = (A^{-1})^n

✅ Rule for Negative Matrix Powers

For a square matrix **A**:

$$A^{-n} = (A^{-1})^n$$

This is only possible if **A is invertible** (its determinant ≠ 0).

If the matrix is **not invertible**, negative power is **not defined**.

Which means:

$$A^{-2} = (A^{-1})^2$$

✅ Manual Concept Breakdown

Example matrix:

$$A = \begin{bmatrix} 4 & 0 \\ 0 & 5 \end{bmatrix}$$

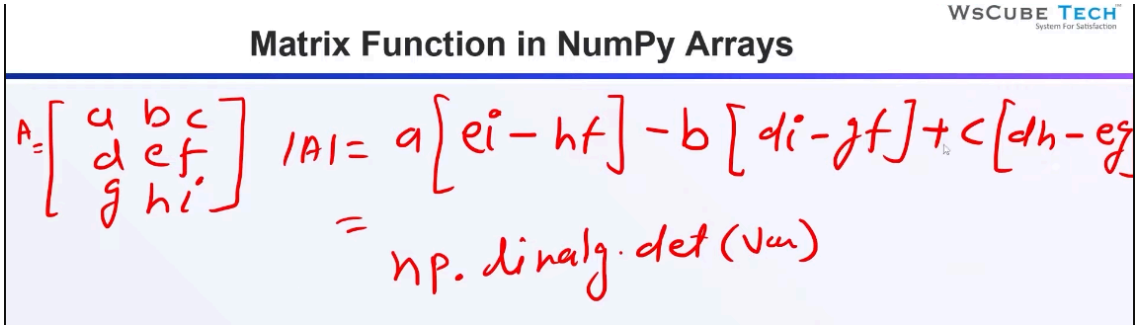
Inverse:

$$A^{-1} = \begin{bmatrix} 1/4 & 0 \\ 0 & 1/5 \end{bmatrix}$$

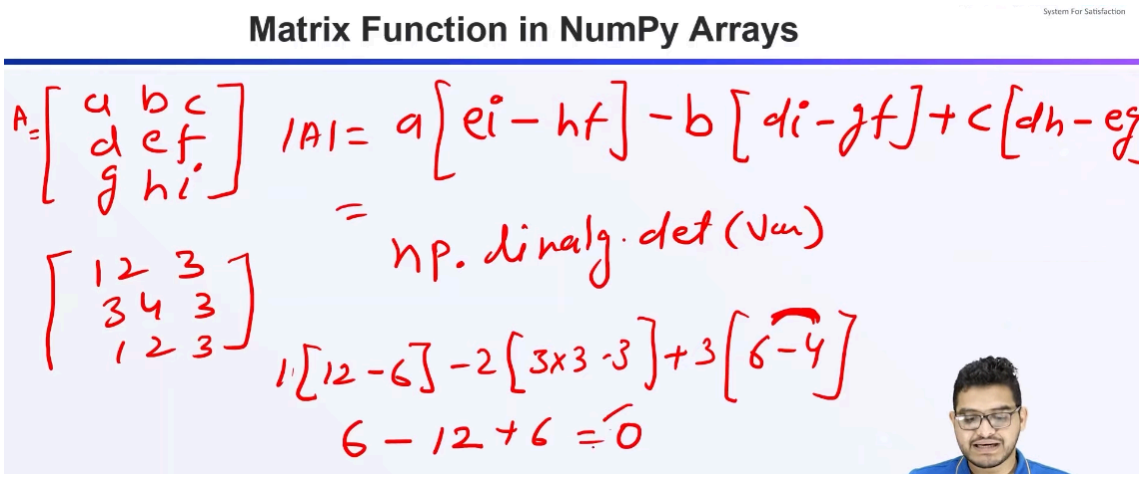
So:

$$A^{-2} = (A^{-1})^2 = \begin{bmatrix} (1/4)^2 & 0 \\ 0 & (1/5)^2 \end{bmatrix}$$

5) Determine

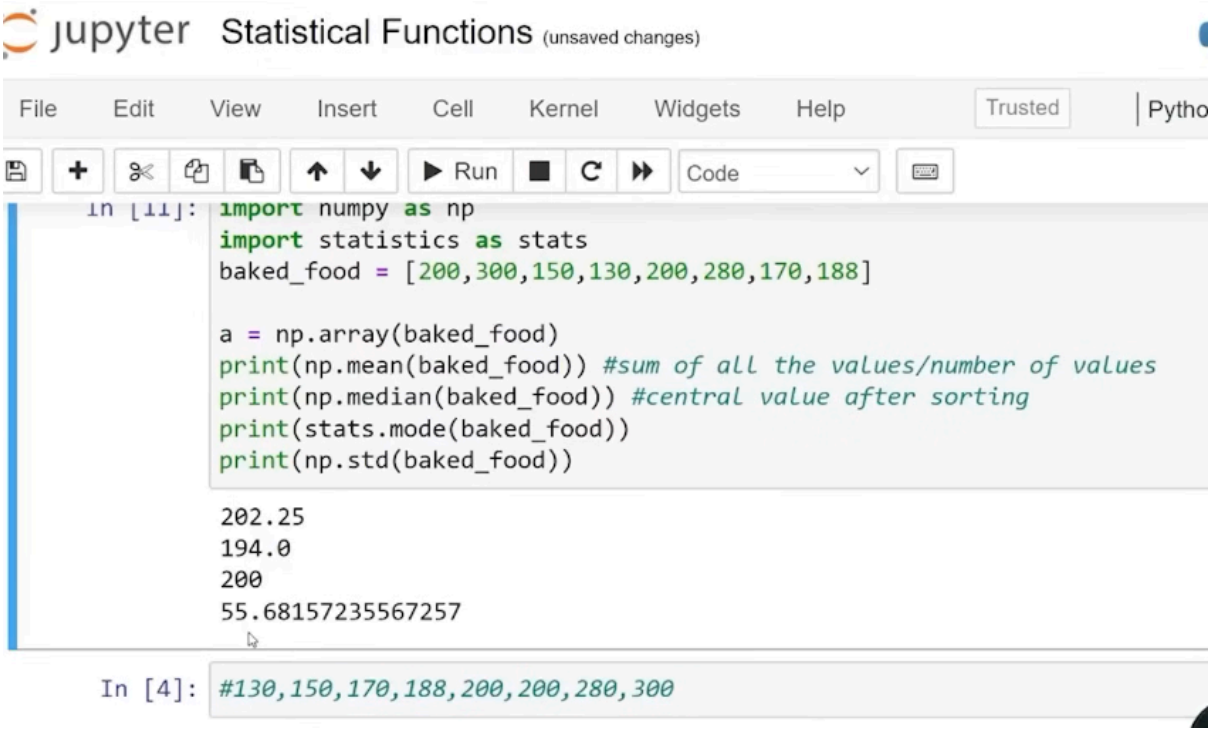



```
var_det = np.matrix([[1,2,3],[4,-5,-6],[7,8,-9]])
print(var_det,"\n")
print(np.linalg.det(var_det))

Output -
[[ 1  2  3]
 [ 4 -5 -6]
 [ 7  8 -9]]

282.00000000000006

Eg 2 -
var_det1 = np.matrix([[1,2,3],[3,4,3],[1,2,3]])
print(var_det1,"\n")
print(np.linalg.det(var_det1))
Output -
[[1 2 3]
 [3 4 3]
 [1 2 3]]

0.0
```


System For Satisfaction
Matrix Function in NumPy Arrays
A = [a b c / d e f / g h i]
|A| = a[ei − hf] − b[di − gf] + c[dh − eg
= np.linalg.det(var)
[1 2 3 / 3 4 3 / 1 2 3]
1[12 − 6] − 2[3x3 − 3] + 3[6 − 4]
6 − 12 + 6 = 0

statistical f'n

Statistical Functions



print(np.var(baked_food)) #basically variance is square of Standard deviations
Output - 842…

Coefficient of Correlations

```
In [17]: # - 1 represent inversely proportional relationship
         # 1 represents proportional relationship
         # 0 means no relationship
         tobacco_consumption = [30,50,10,30,50,40]
         deaths = [100,120,70,100,120,112]

         print(np.corrcoef([tobacco_consumption,deaths]))
```

```
[[1.         0.99015454]
 [0.99015454 1.        ]]
```

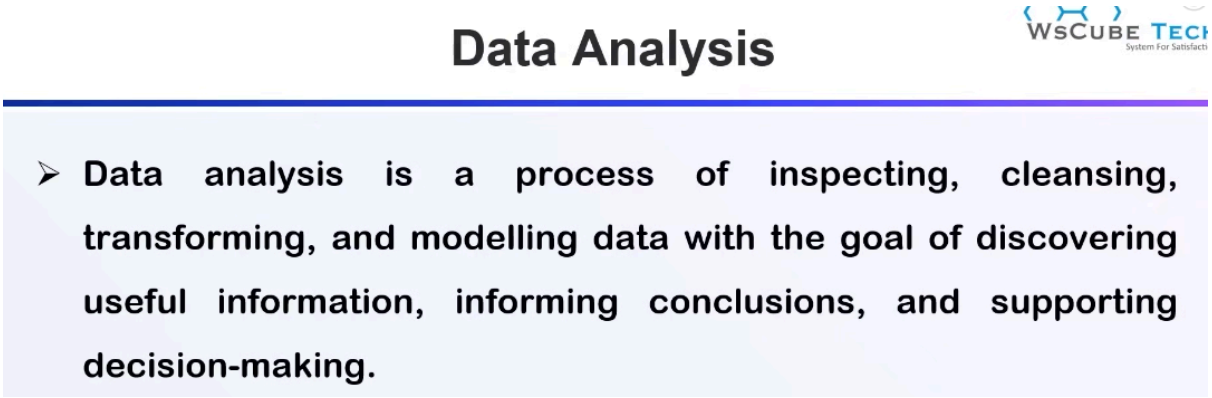
```
In [18]: price = [300,100,350,150,200]
         sales = [10,20,7,17,3]

         print(np.corrcoef([price,sales]))
```

```
[[ 1.         -0.66621445]
 [-0.66621445  1.        ]]
```

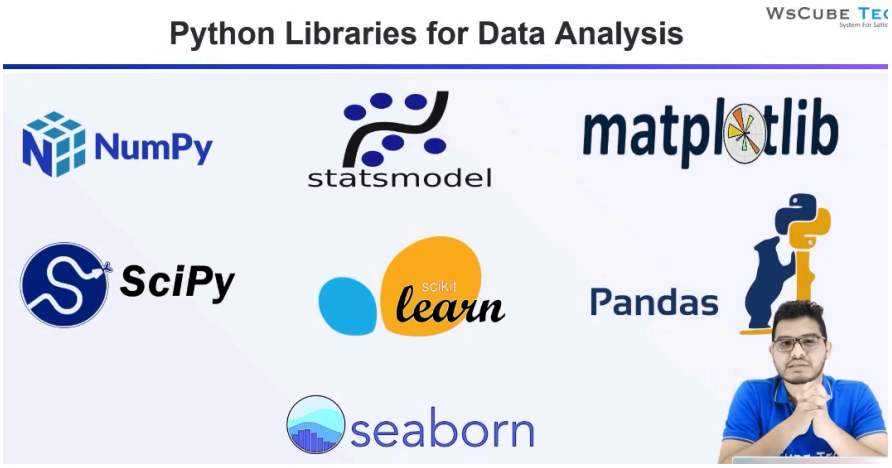
Pandas

Introduction

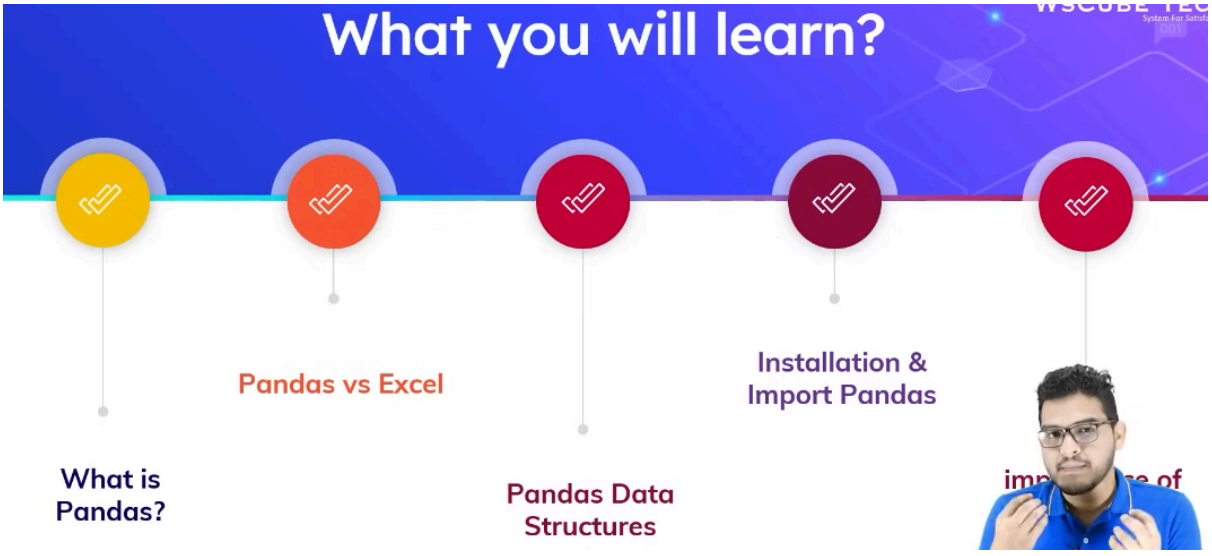
1) Data Analysis


Data Analysis
WsCube Tech
Data analysis is a process of inspecting, cleansing, transforming, and modelling data with the goal of discovering useful information, informing conclusions, and supporting decision-making.

2) Library for Data Analysis



3) Pandas



What is Pandas?

- The name "pandas" has a reference to both "panel data", and "python data analysis" and was created by wes mckinney in 2008.
- Pandas is a python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Read and write data structures and different formats : csv, XML, JSON,ZIP etc.

We need pandas, cause filter or many functions in excel takes lots of time when we have large amount of data like (2 lac+ data) , but yeah if we have data only in thousands so excel will work smoothly good…

Pandas vs Excel

	A	B	C	D
1	RowID	Variable_1	Variable_2	Variable_3
2	1	12.34	aa	12
3	2	34	aa	33
4	3	44	cc	#N/A
5	4	-433	ff	44
6	5		gg	66
7	6	43	dd	33
8	7	34	#NAME?	66
9	8	6743	df	22
10	9	3	fg	
11	10	4	dd	88
12	11	3	g	55
13	12	#N/A	dd	79

4) Pandas Data Structure.

Pandas Data Structures

WsCube Tech

Three Data Structures:

Series: One-Dimensional labeled arrays pd.Series(data)

DataFrames: Two-Dimensional data structure with columns, much like a table.

Panel : A panel is a 3D container of data.

Series 1

	Mango
0	4
1	5
2	6
3	3
4	1

+

Series 2

	Apple
0	5
1	4
2	3
3	0
4	2

=

DataFrame

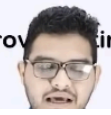
	Mango	Apple
0	4	5
1	5	4
2	6	3
3	3	0
4	1	2

5) Importance

Importance of Pandas in Python

WsCube Tech

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Data set merging and joining. Flexible reshaping and pivoting of data sets Prov ime-series functionality.



6) Type...

Install & import

Install & Import Pandas

1) Intro

INSTALLATION
pip install pandas
Command Prompt
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.
C:\Users\WSCUBETECH>pip install pandas
Requirement already satisfied: pandas in c:\python310\lib\site-packages (1.3.5)
Requirement already satisfied: pytz>=2017.3 in c:\python310\lib\site-packages (from pandas) (2021.3)
Requirement already satisfied: numpy>=1.21.0 in c:\python310\lib\site-packages (from pandas) (1.21.4)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\python310\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\python310\lib\site-packages (from python-dateutil>=2.7.3->pandas)
C:\Users\WSCUBETECH>

2) Same for Numpy

Like -
For Installations - pip install numpy
For Import - import numpy as np

3) Type...

DS (Series) in pd

Data Structure in Pandas

1) Intro

Data Structure in Python Pandas

WSCUBE TECH

- The Pandas provides two data structures for processing the data.

Series & DataFrame and Panel

2) Series Data Structure

- **Series :** It is defined as a one-dimensional array that is capable of storing various data types.

```
import pandas as pd
a = pd.Series( )
print(a)
```

Eg 1 -

```
x = [2,12,4,3,7,9]

var = pd.Series(x)
print(var,"\n")
print(type(var),"\n")
print(var[2])
```

```
Output -
0      2
1     12
2      4
3      3
4      7
5      9
dtype: int64

<class 'pandas.core.series.Series'>

4
```

Eg 2 - (we can also give our index number, data type, name)

```
xy = [2,12,4,3,5]

var = pd.Series(xy,index =
['a','b','s','x','n'],dtype='float',name= 'pandas_python')
print(var)
```

Output -

```
a     2.0
b    12.0
s     4.0
x     3.0
n     5.0
Name: pandas_python, dtype: float64
```

Also can be use in Dict (it's also 2_d)

Eg 3-

```
dic = {"name" :
["python","C++","Java","c"],"por":[12,14,21,9],"Rank":[1,3,2
,4]}

var1 = pd.Series(dic)
print(var1)
```

```
Output -
name     [python, C++, Java, c]
por              [12, 14, 21, 9]
Rank                [1, 3, 2, 4]
dtype: object
```

Eg 4-

```
s = pd.Series(12,index = [1,2,3,4,5,6,7])
print(s)
print(type(s))
```

Output -

```
1    12
2    12
3    12
4    12
5    12
```

```
6    12
7    12
dtype: int64
<class 'pandas.core.series.Series'>
```

Eg 5-

```
s1 = pd.Series(12,index = [1,2,3,4,5,6,7])
s2 = pd.Series(10,index = [1,2,3,4])

print(s1+s2)
```

Output -

```
1    22.0
2    22.0
3    22.0
4    22.0
5     NaN
6     NaN
7     NaN
dtype: float64
```

3) Daf

4)

DS (DataFrame) in pd

DataFrame in Pandas

Intro

In the pandas library for Python, a DataFrame is a two-dimensional, mutable, tabular data structure with labeled axes (rows and columns). It is a fundamental object for data manipulation and analysis in pandas.

1) DataFrame by list

Eg-

```
l = [1,2,3,4,5]

#print(type(l)) -- list
var = pd.DataFrame(l)
print(var,"\n")
print(type(var))
```

Output -

```
   0
0  1
1  2
2  3
3  4
4  5

<class 'pandas.core.frame.DataFrame'>
```

2) DataFrame by Dictionary

Eg -

```
d = {"a":[1,2,3,4,5],"b":[1,2,3,4,5]}
var1 = pd.DataFrame(d)
print(var1)
print(type(var1))
```

Output

```
   a  b
0  1  1
1  2  2
2  3  3
3  4  4
4  5  5
<class 'pandas.core.frame.DataFrame'>
```

Point - in dictionary no of element should be same…

- We can also fetch specific columns and change index no values…

```
dy = {"a":[1,2,3,4,5],"b":[1,2,3,4,5],"c":[1,2,3,4,5],1:[1,2,3,4,5]}
var2 = pd.DataFrame(dy,columns=["a",1],index=["a","b","c","d","e"])
var2 #--like this it will print as dataframe
```

✓ 0.0s

	a	1
a	1	1
b	2	2
c	3	3
d	4	4
e	5	5

- And if we want a specific entity(like (r,c))--> (x - axix,y - axis)

```
dy = {"a":[1,2,3,4,5],"b":[1,2,12,4,5],"c":[1,2,3,4,5],1:[1,2,3,4,5]}
var2 = pd.DataFrame(dy)
print(type(var2))
print(var2,"\n")
print(var2["b"][2]) #--for b 12
```

[19] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
   a   b  c  1
0  1   1  1  1
1  2   2  2  2
2  3  12  3  3
3  4   4  4  4
4  5   5  5  5

12
```

3) DataFrame by Two or more lists

```
list1 = [[1,2,3,4,5],[11,12,13,14,15]]

var3 = pd.DataFrame(list1)
```

```
print(var3)
print(type(var3))
```

```
Output -

   0   1   2   3   4
0   1   2   3   4   5
1  11  12  13  14  15
<class 'pandas.core.frame.DataFrame'>
```

```
Eg 2-
list1 = [[1,2,3,4,5],[11,12,13,14,15],[12,13]]

var3 = pd.DataFrame(list1)
print(var3)
print(type(var3))
```

```
Output -
   0   1     2     3     4
0   1   2   3.0   4.0   5.0
1  11  12  13.0  14.0  15.0
2  12  13   NaN   NaN   NaN
<class 'pandas.core.frame.DataFrame'>
```

4) DataFrame by Two or more Series

```
sr = {"s":pd.Series([1,2,3,4,5]),"r":pd.Series([1,2,3,4,5])}

var4 = pd.DataFrame(sr)
var4
```

✓ 0.0s

	s	r
0	1	1
1	2	2
2	3	3
3	4	4
4	5	5

```
print(type(var4))
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
```

insert data by pandas

Insert data by Pandas

1️⃣ Create the Table using Pandas DataFrame

Step 1: Install libraries (if not installed)

pip install pandas

Step 2: Create the DataFrame

```
import pandas as pd

data = [
    [101, "Rahul", "IT", 45000, "2022-01-15", "Delhi", None],
    [102, "Ankit", "HR", None, "2021-08-10", "Mumbai", None],
    [103, "Neha", "Finance", 52000, None, "Pune", None],
    [104, "Rohan", "IT", 48000, "2023-03-01", None, None],
    [105, "Priya", "Marketing", None, "2020-11-25", "Bangalore", None],
    [None, None, None, None, None, None, None],
    [106, "Aman", "Sales", 40000, "2021-06-18", "Delhi", None],
    [107, "Simran", "HR", 38000, None, "Chandigarh", None],
    [108, "Kunal", "IT", None, "2022-09-05", "Noida", None],
    [109, "Pooja", "Finance", 55000, "2020-02-14", "Mumbai", None],
    [None, None, None, None, None, None, None],
    [110, "Arjun", "Sales", 42000, "2023-01-20", None, None],
    [111, "Sneha", "Marketing", 46000, "2021-04-11", "Jaipur", None],
    [112, "Mohit", "IT", 50000, None, "Gurgaon", None],
    [113, "Nisha", "HR", None, "2019-07-30", "Delhi", None],
    [114, "Akash", "Finance", 60000, "2022-12-01", None, None],
    [115, "Kavya", "Sales", None, None, "Mumbai", None],
    [None, None, None, None, None, None, None],
    [116, "Deepak", "IT", 47000, "2023-05-09", "Pune", None],
    [117, "Ritu", "Marketing", 44000, "2021-10-17", None, None],
    [118, "Saurabh", "HR", None, "2020-03-22", "Noida", None],
    [119, "Isha", "Finance", 53000, None, "Bangalore", None],
]

columns = ["EmployeeID", "Name", "Department", "Salary", "JoinDate", "City", "Bonus"]

df = pd.DataFrame(data, columns=columns)
print(df)
```

✅ **Result:** Table exactly like the one you asked for

- Blank rows → `NaN`
- Bonus column → fully NULL

2️⃣ Insert / Save the Table to CSV or Excel

Save as CSV

df.to_csv("employees_dirty_data.csv", index=False)

Save as Excel

df.to_excel("employees_dirty_data.xlsx", index=False)

📁 You can now:

- Upload CSV/Excel to **Power BI**
- Upload to **GitHub**
- Use for **resume projects**

3️⃣ Insert this Table into a SQL Database (Very Important)

Example: SQLite (Beginner-friendly)

```
import sqlite3

conn = sqlite3.connect("company.db")

df.to_sql(
    name="employees",
    con=conn,
    if_exists="replace",
    index=False
)

conn.close()
```

✅ Table `employees` created with:

- NULL values
- Empty rows
- Null-only column

🔥 Bonus: Check NULLs (Interview-style)

df.isnull().sum()

df[df.isnull().all(axis=1)]

🧠 What you should say in interview:

> “I used Python (Pandas) to insert raw messy data, handled null values, removed empty rows, and then exported cleaned data to Power BI and SQL for analysis.”

If you want next:

- ✅ **Data cleaning using pandas**
- ✅ **Remove null-only columns**
- ✅ **Power BI import steps**
- ✅ **Project-ready Python script**

Just say **next** 🚀

Airthmetic oprn

Arithmetic Operations in Pandas

1) Arithmetic operations
= +,-,*,/ etc…

```
ar_var = pd.DataFrame({"a":[1,2,3,4],"b":[1,2,3,4,]})

ar_var
```
✓ 0.0s

	a	b
0	1	1
1	2	2
2	3	3
3	4	4

```
print(type(ar_var))
```
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
```

2) Addition

```
ar_var["c"] = ar_var["a"] + ar_var["b"]
print(ar_var)
```

```
   a  b  c
0  1  1  2
1  2  2  4
2  3  3  6
3  4  4  8
```

3) Substractions

```
ar_var["c"] = ar_var["a"] - ar_var["b"]
print(ar_var)
```

```
   a  b  c
0  1  1  0
1  2  2  0
2  3  3  0
3  4  4  0
```

4) Multiplication

```
ar_var["c"] = ar_var["a"] * ar_var["b"]
print(ar_var)
```

```
   a  b   c
```

```
0  1  1   1
1  2  2   4
2  3  3   9
3  4  4  16
```

5) Divide

```
ar_var["c"] = ar_var["a"] / ar_var["b"]
print(ar_var)
```

```
   a  b    c
0  1  1  1.0
1  2  2  1.0
2  3  3  1.0
3  4  4  1.0
```

6) Conditionally operations

```
ar_var1 = pd.DataFrame({"a":[10,20,30,40],"b":[15,16,17,18]})
ar_var1
```

[37] ✓ 0.0s

	a	b
0	10	15
1	20	16
2	30	17
3	40	18

```
ar_var1["Crt_col"] = ar_var1["a"] >= 20
ar_var1["New_col"] = ar_var1["b"] <=17
ar_var1
```

[40] ✓ 0.0s

	a	b	Crt_col	New_col
0	10	15	False	True
1	20	16	True	True
2	30	17	True	True
3	40	18	True	False

7) adf

Del, Insert

Delete and Insert

1) Insert

```
id_var = pd.DataFrame({"A":[1,2,3,4,5],"B":[7,8,9,10,12]})
id_var
```

[42] ✓ 0.0s

	A	B
0	1	7
1	2	8
2	3	9
3	4	10
4	5	12

```
id_var.insert(0,"New_col",id_var["B"])
id_var
```

[45] ✓ 0.0s

	New_col	A	B
0	7	1	7
1	8	2	8
2	9	3	9
3	10	4	10
4	12	5	12

Eg -2

```
id_var.insert(2,"python",[11,22,33,44,55])
id_var
```

Output -

```
   New_col  A  python   B
0        7  1      11   7
1        8  2      22   8
2        9  3      33   9
3       10  4      44  10
4       12  5      55  12
```

Eg -3

If i want to enter finite no of entity in columns so i also can do that… here is how…

And also i don't need to write word insert … here is also new way….

```
id_var1 = pd.DataFrame({"A":[1,2,3,4,5],"B":[7,8,9,10,12]})
id_var1
```
✓ 0.0s

	A	B
0	1	7
1	2	8
2	3	9
3	4	10
4	5	12

```
id_var1["slicing_col"] = id_var1["A"][:3]
id_var1
```
✓ 0.0s

	A	B	slicing_col
0	1	7	1.0
1	2	8	2.0
2	3	9	3.0
3	4	10	NaN
4	5	12	NaN

```
id_var1["entr_less_col"] = [12,21,32,None,None]
id_var1
```
✓ 0.0s

	A	B	slicing_col	entr_less_col
0	1	7	1.0	12.0
1	2	8	2.0	21.0
2	3	9	3.0	32.0
3	4	10	NaN	NaN
4	5	12	NaN	NaN

2) Delete

Functions - **pop and del**

DELETE BY - POP AND DEL

```
id_var1
```

[53] ✓ 0.0s

	A	B	slicing_col	entr_less_col
0	1	7	1.0	12.0
1	2	8	2.0	21.0
2	3	9	3.0	32.0
3	4	10	NaN	NaN
4	5	12	NaN	NaN

```
v1 = id_var1.pop("B")
v1 #--- it showing like what it deleted...
```

[54] ✓ 0.0s

```
0     7
1     8
2     9
3    10
4    12
Name: B, dtype: int64
```

```
#it delete --B there for now id_var1 showing this...
id_var1
```

✓ 0.0s

	A	slicing_col	entr_less_col
0	1	1.0	12.0
1	2	2.0	21.0
2	3	3.0	32.0
3	4	NaN	NaN
4	5	NaN	NaN

```
#Now for del
del id_var1["entr_less_col"]
id_var1
```

✓ 0.0s

	A	slicing_col
0	1	1.0
1	2	2.0
2	3	3.0
3	4	NaN
4	5	NaN

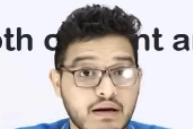
CSV Create and Read

CSV File : Create And Read

1) Difference b/w CSV and EXCEL file.

Differences between CSV and XLS (excel) file formats :

- CSV format is a plain text format in which values are separated by commas (Comma Separated Values).
- XLS file format is an Excel Sheets binary file format which holds information about all the worksheets in a file, including both c nt and formatting.



2) Create Csv file by pandas

```
import pandas as pd
c_var =
pd.DataFrame({"a":[1,2,3,4,5],"b":[1,2,3,4,5],"c":[1,2,3,4,5
]})

c_var
c_var.to_csv("New_crt_file.csv")
```

Output -

	A	B	C	D
1		a	b	c
2	0	1	1	1
3	1	2	2	2
4	2	3	3	3
5	3	4	4	4
6	4	5	5	5
7				

1) **If you don't want to keep index numbers(series.. 0,1,2…4)**

```
c_var.to_csv("New_crt_file1.csv",index=False)
```

Output - (just changed file name as that file already exists)

	A	B	C
1	a	b	c
2	1	1	1
3	2	2	2
4	3	3	3
5	4	4	4
6	5	5	5

2) **And, if you want change header name …**

```
c_var.to_csv("New_crt_file2.csv",index=False,header=["
new1","new2","new3"])
```

Output -

A	B	C
new1	new2	new3
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5

3) Read CSV file by Pandas

CSV Files :

- A simple way to store big data sets is to use CSV files (comma separated files).
- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

1) Read or open file (pd.read_csv("location+name") – use \\ if nt wrk

```
R_var =
pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd
w Hry\\Python_library\\Pandas\\New_crt_file.csv")
R_var
```

Output -

```
R_var = pd.read_csv("C:\
R_var
```

✓ 0.0s

	Unnamed: 0	a	b	c
0	0	1	1	1
1	1	2	2	2
2	2	3	3	3
3	3	4	4	4
4	4	5	5	5

2) If you want to read specific no of lines … (nrows = _)

```
#for only 1st rows
R_var1 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\New_crt_file.csv",nrows=1)
R_var1
```

✓ 0.0s

	Unnamed: 0	a	b	c
0	0	1	1	1

```
#for only first 3 rows
R_var1 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\New_crt_file.csv",nrows=3)
R_var1
```

✓ 0.0s

	Unnamed: 0	a	b	c
0	0	1	1	1
1	1	2	2	2
2	2	3	3	3

3) If want specific columns.. (uscols = [])

```
#for only specific columns by Name
R_var2 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\New_crt_file.csv",usecols=["a","c"])
R_var2
```

22] ✓ 0.0s

	a	c
0	1	1
1	2	2
2	3	3
3	4	4
4	5	5

```
#for only specific columns by index number
R_var2 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\New_crt_file.csv",nrows=2,usecols=[0,2])
R_var2
```

24] ✓ 0.0s

	Unnamed: 0	b
0	0	1
1	1	2

4) Skip rows…

```
R_var
```

26] ✓ 0.0s

	Unnamed: 0	a	b	c
0	0	1	1	1
1	1	2	2	2
2	2	3	3	3
3	3	4	4	4
4	4	5	5	5

```
#for skip specific rows -- single
R_var3 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\New_crt_file.csv",skiprows=2)
R_var3
```

32] ✓ 0.0s

	1	2	2.1	2.2
0	2	3	3	3
1	3	4	4	4
2	4	5	5	5

```
#for skip specific rows -- single
R_var3 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\New_crt_file.csv",skiprows=4)
R_var3
```

33] ✓ 0.0s

	3	4	4.1	4.2
0	4	5	5	5

5) Now, we can also make a column as index column

```
R_var4 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\New_crt_file.csv",index_col="b")
R_var4
```

✓ 0.0s

b	Unnamed: 0	a	c
1	0	1	1
2	1	2	2
3	2	3	3
4	3	4	4
5	4	5	5

6) Also, we can make any row as header..

```
R_var5 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\New_crt_file.csv",header = 2)
R_var5
```

✓ 0.0s

	1	2	2.1	2.2
0	2	3	3	3
1	3	4	4	4
2	4	5	5	5

- **As entity in rows are same like (1,2,2,2 - but columns name can't be same that's why header name updated as - 1,2,2.1,2.2)**

```
R_var6 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\chatgpt_data1.csv",header=2)
R_var6
#but which header we keep, it removes all the upper rows data till that (2nd) row.
```

✓ 0.0s

	2	Text_2	4	3	Item_2
0	3	Text_3	6	4.5	Item_3
1	4	Text_4	8	6.0	Item_4
2	5	Text_5	10	7.5	Item_5
3	6	Text_6	12	9.0	Item_6
4	7	Text_7	14	10.5	Item_7
5	8	Text_8	16	12.0	Item_8

7) **We can also change heading and give name by ourselves**

```
R_var6 =
pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd
w Hry\\Python_library\\Pandas\\chatgpt_data1.csv",names=
["col"])
R_var6
```

Output

				col
Column1	Column2	Column3	Column4	Column5
1	Text_1	2	1.5	Item_1
2	Text_2	4	3	Item_2
3	Text_3	6	4.5	Item_3

so on...

If you want change all the col name so...

```
R_var7 =
pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd
w Hry\\Python_library\\Pandas\\chatgpt_data1.csv",names=
["col1","col2","col3","col4","col5"],nrows=4)
R_var7
```

```
R_var7 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\chatgpt_data1.csv",names= ["col1","col2","col3"
R_var7
```

✓ 0.0s Python

	col1	col2	col3	col4	col5
0	Column1	Column2	Column3	Column4	Column5
1	1	Text_1	2	1.5	Item_1
2	2	Text_2	4	3	Item_2
3	3	Text_3	6	4.5	Item_3

8) **What if we have to add header with another name and use this existed header as row...**

```
R_var9 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\chatgpt_data1.csv")
R_var9
```

✓ 0.0s

	1	Text_1	2	1.5	Item_1
0	2	Text_2	4	3.0	Item_2
1	3	Text_3	6	4.5	Item_3
2	4	Text_4	8	6.0	Item_4

So -

```
R_var10 =
pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd
w_Hry\\Python_library\\Pandas\\chatgpt_data1.csv",header=Non
e, prefix= "col")
R_var10
```

Or

```
R_var10 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\chatgpt_data1.csv",header=None)
R_var10
```

✓ 0.0s

	0	1	2	3	4
0	1	Text_1	2	1.5	Item_1
1	2	Text_2	4	3.0	Item_2
2	3	Text_3	6	4.5	Item_3
3	4	Text_4	8	6.0	Item_4
4	5	Text_5	10	7.5	Item_5
5	6	Text_6	12	9.0	Item_6

After it i can change name like -

1) Save this file as header none or removed that rows as header

```
R_var11 = R_var10.to_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\chatgpt_data1.csv")
R_var11 = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd w Hry\\Python_library\\Pandas\\chatgpt_data1.csv")
R_var11
```

✓ 0.0s

	Unnamed: 0	0	1	2	3	4
0	0	1	Text_1	2	1.5	Item_1
1	1	2	Text_2	4	3.0	Item_2
2	2	3	Text_3	6	4.5	Item_3
3	3	4	Text_4	8	6.0	Item_4

2) And then now i will give it new name…

```
R_var11 =
R_var10.to_csv("C:\\Users\\DELL\\OneDrive\\Documents\\
Python_cd w
Hry\\Python_library\\Pandas\\chatgpt_data1.csv")

R_var11 =
pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Pyt
hon_cd w
Hry\\Python_library\\Pandas\\chatgpt_data1.csv",names=
["col1","col2","col3","col4","col5"])

R_var11
```

	col1	col2	col3	col4	col5
NaN	0	1	2	3.0	4
0.0	1	Text_1	2	1.5	Item_1
1.0	2	Text_2	4	3.0	Item_2

9) **If i want to change data type of any columns -**

```
R_var12 =

pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd
w
Hry\\Python_library\\Pandas\\chatgpt_data1.csv",dtype={"col3
":"float","col1":"float"})

R_var12
```

```
= pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents\\Python_cd
```

0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
0	0	1.0	Text_1	2.0	1.5	Item_1
1	1	2.0	Text_2	4.0	3.0	Item_2
2	2	3.0	Text_3	6.0	4.5	Item_3
3	3	4.0	Text_4	8.0	6.0	Item_4
4	4	5.0	Text_5	10.0	7.5	Item_5
5	5	6.0	Text_6	12.0	9.0	Item_6
6	6	7.0	Text_7	14.0	10.5	Item_7

functions in csv files

Functions of CSV File

1) Normal table -

```
cf_var = pd.read_csv("C:\\Users\\DELL\\OneDrive\\Documents
cf_var
```

✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
0	0	1	Text_1	2	1.5	Item_1
1	1	2	Text_2	4	3.0	Item_2
2	2	3	Text_3	6	4.5	Item_3
3	3	4	Text_4	8	6.0	Item_4
4	4	5	Text_5	10	7.5	Item_5
5	5	6	Text_6	12	9.0	Item_6
6	6	7	Text_7	14	10.5	Item_7
7	7	8	Text_8	16	12.0	Item_8
8	8	9	Text_9	18	13.5	Item_9
9	9	10	Text_10	20	15.0	Item_10
10	10	11	Text_11	22	16.5	Item_11
11	11	12	Text_12	24	18.0	Item_12
12	12	13	Text 13	26	19.5	Item 13

2) Check index range - (csv.index)

```
Cf_var.index
```

```
Output -
RangeIndex(start=0, stop=25, step=1)
```

3) Check total columns - (csv.columns)

```
Cf_var.columns
```

```
Output -
Index(['Unnamed: 0', 'col1', 'col2', 'col3', 'col4',
'col5'], dtype='object')
```

4) Descriptions of table_ special for int columns…(csv.describe())

```
#Descriptions of table_ special for int columns…(csv.describe)
cf_var.describe()
```

✓ 0.0s

	Unnamed: 0	col1	col3	col4
count	25.000000	25.000000	25.000000	25.000000
mean	12.000000	13.000000	26.000000	19.500000
std	7.359801	7.359801	14.719601	11.039701
min	0.000000	1.000000	2.000000	1.500000
25%	6.000000	7.000000	14.000000	10.500000
50%	12.000000	13.000000	26.000000	19.500000
75%	18.000000	19.000000	38.000000	28.500000
max	24.000000	25.000000	50.000000	37.500000

5) Fetch specific no (default - 5) from top or head (csv.head())

```
#default
cf_var.head()
```

✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
0	0	1	Text_1	2	1.5	Item_1
1	1	2	Text_2	4	3.0	Item_2
2	2	3	Text_3	6	4.5	Item_3
3	3	4	Text_4	8	6.0	Item_4
4	4	5	Text_5	10	7.5	Item_5

```
cf_var.head(2)
```

✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
0	0	1	Text_1	2	1.5	Item_1
1	1	2	Text_2	4	3.0	Item_2

6) Fetch specific no (default - 5) from bottom or tail (csv.tail())

```
#default
cf_var.tail()
```

✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
20	20	21	Text_21	42	31.5	Item_21
21	21	22	Text_22	44	33.0	Item_22
22	22	23	Text_23	46	34.5	Item_23
23	23	24	Text_24	48	36.0	Item_24
24	24	25	Text_25	50	37.5	Item_25

```
cf_var.tail(2)
```

✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
23	23	24	Text_24	48	36.0	Item_24
24	24	25	Text_25	50	37.5	Item_25

7) Slicing : Fetching rows… (csv[:]) — accordingly

```
cf_var[ :2]
```

12] ✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
0	0	1	Text_1	2	1.5	Item_1
1	1	2	Text_2	4	3.0	Item_2

```
cf_var[3:7] #for last row mind - n+1
```
13] ✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
3	3	4	Text_4	8	6.0	Item_4
4	4	5	Text_5	10	7.5	Item_5
5	5	6	Text_6	12	9.0	Item_6
6	6	7	Text_7	14	10.5	Item_7

```
cf_var[22:]
```
14] ✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
22	22	23	Text_23	46	34.5	Item_23
23	23	24	Text_24	48	36.0	Item_24
24	24	25	Text_25	50	37.5	Item_25

8) Type functions (print(type(csv))))

```
print(type(cf_var))
```

```
Output -
<class 'pandas.core.frame.DataFrame'>
```

9) Get index array – (csv.index.array)

```
Cf_var.index.array
```

```
Output -
<NumpyExtensionArray>
[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
15, 16, 17, 18,
 19, 20, 21, 22, 23, 24]
Length: 25, dtype: int64
```

10) Convert csv to Numpy array -

Two methods —

a) Without import Numpy – (csv.to_numpy())

```
cf_var.to_numpy()
```

```
cf_var.to_numpy()
```
✓ 0.0s

```
array([[0, 1, 'Text_1', 2, 1.5, 'Item_1'],
       [1, 2, 'Text_2', 4, 3.0, 'Item_2'],
       [2, 3, 'Text_3', 6, 4.5, 'Item_3'],
       [3, 4, 'Text_4', 8, 6.0, 'Item_4'],
       [4, 5, 'Text_5', 10, 7.5, 'Item_5'],
       [5, 6, 'Text_6', 12, 9.0, 'Item_6'],
```

```
        [5, 6, 'Text_6', 12, 9.0, 'Item_6'],
        [6, 7, 'Text_7', 14, 10.5, 'Item_7'],
        [7, 8, 'Text_8', 16, 12.0, 'Item_8'],
        [8, 9, 'Text_9', 18, 13.5, 'Item_9'],
        [9, 10, 'Text_10', 20, 15.0, 'Item_10'],
        [10, 11, 'Text_11', 22, 16.5, 'Item_11'],
        [11, 12, 'Text_12', 24, 18.0, 'Item_12'],
        [12, 13, 'Text_13', 26, 19.5, 'Item_13'],
        [13, 14, 'Text_14', 28, 21.0, 'Item_14'],
        [14, 15, 'Text_15', 30, 22.5, 'Item_15'],
        [15, 16, 'Text_16', 32, 24.0, 'Item_16'],
        [16, 17, 'Text_17', 34, 25.5, 'Item_17'],
        [17, 18, 'Text_18', 36, 27.0, 'Item_18'],
        [18, 19, 'Text_19', 38, 28.5, 'Item_19'],
        [19, 20, 'Text_20', 40, 30.0, 'Item_20'],
        [20, 21, 'Text_21', 42, 31.5, 'Item_21'],
        [21, 22, 'Text_22', 44, 33.0, 'Item_22'],
        [22, 23, 'Text_23', 46, 34.5, 'Item_23'],
        [23, 24, 'Text_24', 48, 36.0, 'Item_24'],
        [24, 25, 'Text_25', 50, 37.5, 'Item_25']], dtype=object)
```

b) By import numpy –

```
import numpy as np

ncf_var = np.array(cf_var)
ncf_var
```

Output -

```
cf_var.to_numpy()
```
✓ 0.0s

```
array([[0, 1, 'Text_1', 2, 1.5, 'Item_1'],
       [1, 2, 'Text_2', 4, 3.0, 'Item_2'],
       [2, 3, 'Text_3', 6, 4.5, 'Item_3'],
       [3, 4, 'Text_4', 8, 6.0, 'Item_4'],
       [4, 5, 'Text_5', 10, 7.5, 'Item_5'],
       [5, 6, 'Text_6', 12, 9.0, 'Item_6'],
```

Same as above…

11) Change index in descending

Axis - 0 = rows, | Axis - 1 = columns

```
#Change index in descending
cf_var.sort_index(axis=0,ascending=False)
```
✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
24	24	25	Text_25	50	37.5	Item_25
23	23	24	Text_24	48	36.0	Item_24
22	22	23	Text_23	46	34.5	Item_23
21	21	22	Text_22	44	33.0	Item_22

```
cf_var.sort_index(axis=1,ascending=False)
```

[21] ✓ 0.0s

	col5	col4	col3	col2	col1	Unnamed: 0
0	Item_1	1.5	2	Text_1	1	0
1	Item_2	3.0	4	Text_2	2	1
2	Item_3	4.5	6	Text_3	3	2

12) Now, change Specific data

```
#find specific data
Cf_var["col3"][2] # column then Row
```

Output -

6

Now i change this - by 33

```
#change specific data
cf_var["col3"][2] = 33
Cf_var
```

Output -

```
#change specific data
cf_var["col3"][2] = 33
cf_var
```

✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_20460\4041963276.py:
You are setting values through chained assignment. Currently th
A typical example is when you are setting values in a column of

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform t

See the caveats in the documentation: https://pandas.pydata.org

  cf_var["col3"][2] = 33
C:\Users\DELL\AppData\Local\Temp\ipykernel_20460\4041963276.py:
A value is trying to be set on a copy of a slice from a DataFra

See the caveats in the documentation: https://pandas.pydata.org
  cf_var["col3"][2] = 33
```

	Unnamed: 0	col1	col2	col3	col4	col5
0	0	1	Text_1	2	1.5	Item_1
1	1	2	Text_2	4	3.0	Item_2
2	2	3	Text_3	33	4.5	Item_3

it change but it also give error… so this is not a efficient way…

Efficient way to do this … without any error – use loc function.

```
cf_var.loc[1,"col3"] = 22 #--Now, Row-columns
cf_var
```

	Unnamed: 0	col1	col2	col3	col4	col5
0	0	1	Text_1	2	1.5	Item_1
1	1	2	Text_2	22	3.0	Item_2
2	2	3	Text_3	33	4.5	Item_3

13) Find specific data by rows and columns

```
cf_var.loc[[3,5],["col3","col5"]]
```
✓ 0.0s

	col3	col5
3	8	Item_4
5	12	Item_6

For full row and specific columns –

```
cf_var.loc[:,["col3","col5"]]
```
✓ 0.0s

	col3	col5
0	2	Item_1
1	22	Item_2
2	33	Item 3

so on till 25…

For full columns and specific rows –

```
cf_var.loc[[2,3],:]
```
✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
2	2	3	Text_3	33	4.5	Item_3
3	3	4	Text_4	8	6.0	Item_4

14) Again getting a specific data by using - iloc (but this is by index no)

```
cf_var.iloc[2,2] #-- row-column
#finding - Text_3

Output -
'Text_3'

cf_var.iloc[0,4] #-- row-column
#finding - 1.5
Output - 1.5
```

15) Deleting or Dropping any Row or Column…

Dropping rows - 0 rows

```
# Axis = 0 - row | Axis = 1 - Column
cf_var.drop(0,axis=0)
```

✓ 0.0s

	Unnamed: 0	col1	col2	col3	col4	col5
1	1	2	Text_2	22	3.0	Item_2
2	2	3	Text_3	33	4.5	Item_3
3	3	4	Text_4	8	6.0	Item_4

Dropping columns - col3

```
# Axis = 0 - row | Axis = 1 - Column
cf_var.drop("col3",axis=1)
```

✓ 0.0s

	Unnamed: 0	col1	col2	col4	col5
0	0	1	Text_1	1.5	Item_1
1	1	2	Text_2	3.0	Item_2
2	2	3	Text_3	4.5	Item_3

16) Note -

This is not deleting any data in original file it's just showing here after modifying according functions…

hndl miss data

Handling Missing Data
Dropna and Fillna

1) Dropna - to drop entity.

1) Drop all null values and shows data - [Dropna()]

- It just showing all data after removing all the null values from each rows… and it just s hows not change in real csv file.
- And it generally (or by **default**) along row wise like - **Dropna(axis = 0)**

```
vd.dropna()
```

✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi
9	109.0	Pooja	Finance	55000.0	2020-02-14	Mumbai
12	111.0	Sneha	Marketing	46000.0	2021-04-11	Jaipur
18	116.0	Deepak	IT	47000.0	2023-05-09	Pune

2) Drop all null values and shows data along columns - [Dropna(axis = 1)]

- So, as no column were without null value. That's why ther is showing only one columns which is index column

```
vd.dropna(axis=1)
```

[29] ✓ 0.0s

```
0
1
2
```

-

3) Now drop all null values of rows which has all null data.

csv.dropna(how = ‘any’)

- Its same as dropna because it removes all na entity rows.

```
vd.dropna(how='any')
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi
9	109.0	Pooja	Finance	55000.0	2020-02-14	Mumbai
12	111.0	Sneha	Marketing	46000.0	2021-04-11	Jaipur
18	116.0	Deepak	IT	47000.0	2023-05-09	Pune

csv.dropna(how = ‘all’)

- Its removes only that rows which have all the null entity.

```
vd.dropna(how= 'all')
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	NaN	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN
4	105.0	Priya	Marketing	NaN	2020-11-25	Bangalore

4) Now drop all none value along column - csv.dropna(subset = [“column name”])

- Its remove all na value of particular column and that column will show without any null value.

```
vd.dropna(subset= ['Salary'])
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
2	103.0	Neha	Finance	52000.0	NaN	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi
7	107.0	Simran	HR	38000.0	NaN	Chandigarh
9	109.0	Pooja	Finance	55000.0	2020-02-14	Mumbai
11	110.0	Arjun	Sales	42000.0	2023-01-20	NaN
12	111.0	Sneha	Marketing	46000.0	2021-04-11	Jaipur
13	112.0	Mohit	IT	50000.0	NaN	Gurgaon
15	114.0	Akash	Finance	60000.0	2022-12-01	NaN
18	116.0	Deepak	IT	47000.0	2023-05-09	Pune
19	117.0	Ritu	Marketing	44000.0	2021-10-17	NaN
21	119.0	Isha	Finance	53000.0	NaN	Bangalore

5) Now, we can drop that row which have a specific no of non - null value. csv.dropna(thresh = "specific no")

- Keeps rows with 2 or more non-null values.
- Drops rows that have fewer than 2 non-null values (i.e., those with 0 or 1 non-null value).

```
vd.dropna(thresh = 2)
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	NaN	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN
4	105.0	Priya	Marketing	NaN	2020-11-25	Bangalore

-

```
vd.dropna(thresh=6)
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi
9	109.0	Pooja	Finance	55000.0	2020-02-14	Mumbai
12	111.0	Sneha	Marketing	46000.0	2021-04-11	Jaipur
18	116.0	Deepak	IT	47000.0	2023-05-09	Pune

```
vd.dropna(thresh = 7)
```
✓ 0.0s

EmployeeID	Name	Department	Salary	JoinDate	City

-

- **Thresh = 6 and 7 (alteast 6,7 entity should be non null value)**

2) fillna- to fill blank(null) entity.

1) Fill something at empty place. csv.fillna("xyz")

- Fill anything which you want at empty place.

```
vd.fillna("python")
```

✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	python	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	python	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	python

2) Fill different things at different columns csv.fillna({dictionrary})

- Fill different different things at different different columns.

```
vd.fillna({"EmployeeID":100,"Name":"Default","Salary":"0"})
```

✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	0	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	NaN	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN
4	105.0	Priya	Marketing	0	2020-11-25	Bangalore
5	100.0	Default	NaN	0	NaN	NaN
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi
7	107.0	Simran	HR	38000.0	NaN	Chandigarh

3) Method fillna : fill same as neighbour

Default -

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	NaN	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN
4	105.0	Priya	Marketing	NaN	2020-11-25	Bangalore
5	NaN	NaN	NaN	NaN	NaN	NaN
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi

a) Axis 1 or according rows

i) csv.fillna(method="ffill",axis = 1) - fill same as left entity of rows.

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	HR	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	52000.0	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	2023-03-01
4	105.0	Priya	Marketing	Marketing	2020-11-25	Bangalore
5	NaN	NaN	NaN	NaN	NaN	NaN
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi

ii) csv.fillna(method="bfill",axis = 1) - fill same as right entity of rows.

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	2021-08-10	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	Pune	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN
4	105.0	Priya	Marketing	2020-11-25	2020-11-25	Bangalore
5	NaN	NaN	NaN	NaN	NaN	NaN
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi

b) Axis 0 or default or according column

i) csv.fillna(method="ffill",axis = 0) - fill same as upper entity of columns.

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	45000.0	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	2021-08-10	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	Pune
4	105.0	Priya	Marketing	48000.0	2020-11-25	Bangalore
5	105.0	Priya	Marketing	48000.0	2020-11-25	Bangalore
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi

ii) **csv.fillna(method="bfill") - fill same as lower entity of columns.**

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	52000.0	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	2023-03-01	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	Bangalore
4	105.0	Priya	Marketing	40000.0	2020-11-25	Bangalore
5	106.0	Aman	Sales	40000.0	2021-06-18	Delhi
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi

4) Fill any value in place of null entity.

- But it will change vd(or variable which you were using) but not at csv file . **vd.fillna("Any",inplace = True)**

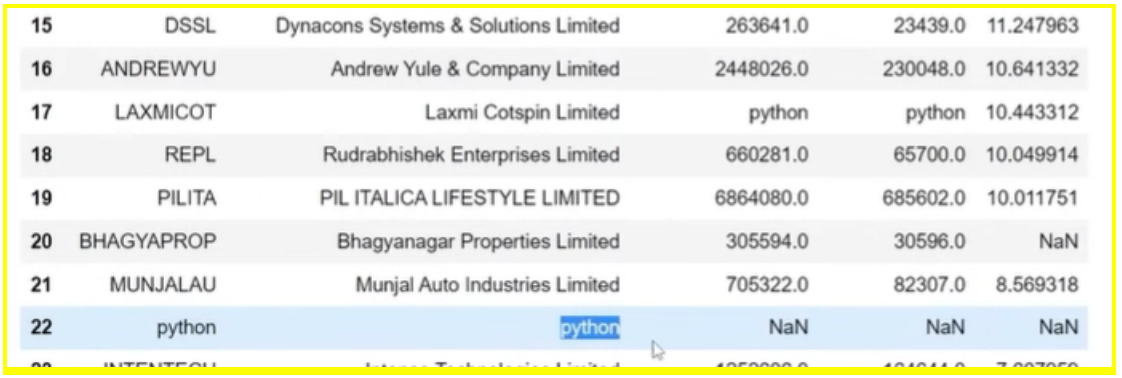
```
vd.fillna("no",inplace=True)
vd
```

✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi
1	102.0	Ankit	HR	no	2021-08-10	Mumbai
2	103.0	Neha	Finance	52000.0	no	Pune
3	104.0	Rohan	IT	48000.0	2023-03-01	no
4	105.0	Priya	Marketing	no	2020-11-25	Bangalore
5	no	no	no	no	no	no
6	106.0	Aman	Sales	40000.0	2021-06-18	Delhi

-

5) it will replace the first number of consecutive missing (NaN, NA, or None) values in the DataFrame or Series vd with the any thing which you want.
vd.fillna("python",limit = 2)



15	DSSL	Dynacons Systems & Solutions Limited	263641.0	23439.0	11.247963
16	ANDREWYU	Andrew Yule & Company Limited	2448026.0	230048.0	10.641332
17	LAXMICOT	Laxmi Cotspin Limited	python	python	10.443312
18	REPL	Rudrabhishek Enterprises Limited	660281.0	65700.0	10.049914
19	PILITA	PIL ITALICA LIFESTYLE LIMITED	6864080.0	685602.0	10.011751
20	BHAGYAPROP	Bhagyanagar Properties Limited	305594.0	30596.0	NaN
21	MUNJALAU	Munjal Auto Industries Limited	705322.0	82307.0	8.569318
22	python	python	NaN	NaN	NaN

hndlmsd(rep.intrpl)

Handling Missing Data
Replace and Interpolate

1) Replace - replace entity...

1) Replace any data with any str of num

(fl.replace(to_replace="IT",value= 0)

```
fl.replace(to_replace="IT",value=0)
```

✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	0	45000.0	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.0	NaN	Pune	NaN
3	104.0	Rohan	0	48000.0	2023-03-01	NaN	NaN

If you want replace many entity with same or different data.

```
fl.replace([101,102,103,104,105,106],42)
```

✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	42.0	Rahul	IT	45000.0	2022-01-15	Delhi	NaN
1	42.0	Ankit	HR	NaN	2021-08-10	Mumbai	NaN
2	42.0	Neha	Finance	52000.0	NaN	Pune	NaN
3	42.0	Rohan	IT	48000.0	2023-03-01	NaN	NaN
4	42.0	Priya	Marketing	NaN	2020-11-25	Bangalore	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	42.0	Aman	Sales	40000.0	2021-06-18	Delhi	NaN

2) But if you want to replace a series like (A to Z or a to z) with any data. Use regex = True

```
fl.replace("[A-Za-z]","PYT",regex=True)
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	PYTPYTPYTPYTPYT	PYTPYT	45000.0	2022-01-15	PYTPYTPYTPYTPYT	NaN
1	102.0	PYTPYTPYTPYTPYT	PYTPYT	NaN	2021-08-10	PYTPYTPYTPYTPYTPYT	NaN
2	103.0	PYTPYTPYTPYT	PYTPYTPYTPYTPYTPYTPYT	52000.0	NaN	PYTPYTPYTPYT	NaN
3	104.0	PYTPYTPYTPYTPYT	PYTPYT	48000.0	2023-03-01	NaN	NaN

```
fl.replace("[a-z]","PY",regex= True)
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	RPYPYPYPY	IT	45000.0	2022-01-15	DPYPYPYPY	NaN
1	102.0	APYPYPYPY	HR	NaN	2021-08-10	MPYPYPYPYPY	NaN
2	103.0	NPYPYPY	FPYPYPYPYPYPY	52000.0	NaN	PPYPYPY	NaN
3	104.0	RPYPYPYPY	IT	48000.0	2023-03-01	NaN	NaN
4	105.0	PPYPYPYPY	MPYPYPYPYPYPYPYPY	NaN	2020-11-25	BPYPYPYPYPYPYPYPY	NaN

And also, we can replace data of a particular column. By use of dictionary

```
fl.replace({"Name":["Rahul","Ankit","Neha"]},"RAN",regex=True)
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	RAN	IT	45000.0	2022-01-15	Delhi	NaN
1	102.0	RAN	HR	NaN	2021-08-10	Mumbai	NaN
2	103.0	RAN	Finance	52000.0	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN	NaN

3) Now we can also replace a number same as their nieghbours. Method - back fill bfill or front fill ffill. Default…

```
fl
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.0	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	NaN	2020-11-25	Bangalore	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1) Front fill- fill along column same as above entity.

fl.replace("replace value",method="ffill")

```
fl.replace("IT",method="ffill")
```
✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_6048\2604261748.py:1: FutureWarn
  fl.replace("IT",method="ffill")
```

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.0	NaN	Pune	NaN
3	104.0	Rohan	Finance	48000.0	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	NaN	2020-11-25	Bangalore	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN

2) Below fill- fill along column same as below entity.

```
fl.replace("IT",method="bfill")
```
✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_6048\4284450191.py:1: FutureWarni
  fl.replace("IT",method="bfill")
```

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	HR	45000.0	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.0	NaN	Pune	NaN
3	104.0	Rohan	Marketing	48000.0	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	NaN	2020-11-25	Bangalore	NaN

3) **And we can also replace value by limit but conditions is … value should be consecutive.**
Default -

5	6	22	Alphageo (India) Limited	1	8356	27.852613
6	7	22	Uniphos Enterprises Limited	1	8357	21.528002
7	8	22	Jagsonpal Pharmaceuticals Limited	1	73134	17.801420
8	9	22	SKM Egg Products Export (India) Limited	1	87775	18.801420
9	10	22	Swelect Energy Systems Limited	1	87776	16.938733
10	11	22	Nahar Poly Films Limited	1	38574	15.555297

fl.replace("value",method="bfill"or"ffill",limit= 1or more)

a) **var.replace("1",method="ffill",limit = 2)**

3	4	GKWLIMITED	GKW Limited	21375	711	30.049203
4	5	21STCENMGM	21st Century Management Services Limited	110375	3879	31.049203
5	6	ALPHAGEO	Alphageo (India) Limited	110375	8356	27.852613
6	7	UNIENTER	Uniphos Enterprises Limited	110375	8357	21.528002
7	8	JAGSNPHARM	Jagsonpal Pharmaceuticals Limited	1	73134	17.801420
8	9	SKMEGGPROD	SKM Egg Products Export (India) Limited	1	87775	18.801420
9	10	SWELECTES	Swelect Energy Systems Limited	1	87776	16.938733
10	11	NAHARPOLY	Nahar Poly Films Limited	1	38574	15.555297
11	12	PODDARHOUS	Poddar Housing and Development Limited	1	3012	15.283833

So on… for bfill or fill…

2) Interpolate… – fill only blank or null values…

- If it use default then it will only work for numerical values and fill by any logic like series, avg, below, above any way…
- csv.interpolate()

```
fl.interpolate()
```
✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_6048\1169794338.py:1: FutureWarning:
  fl.interpolate()
```

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.000000	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	48500.000000	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.000000	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.000000	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	45333.333333	2020-11-25	Bangalore	NaN
5	105.5	NaN	NaN	42666.666667	NaN	NaN	NaN

method :{‘linear’, ‘time’, ‘index’, ‘values’, ‘nearest’, ‘zero’, ‘slinear’, ‘quadratic’, ‘cubic’, ‘barycentric’, ‘krogh’, ‘polynomial’, ‘spline’, ‘piecewise_polynomial’, ‘from_derivatives’, ‘pchip’, ‘akima’}

1) Axis = 0, csv.interpolate(method='linear',axis=0),

Its will fill by assuming linear and along with column and its default.

```
fl.interpolate(method='linear',axis=0)
```

✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_6048\2605692331.py:1: FutureWarning: I
  fl.interpolate(method='linear',axis=0)
```

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.000000	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	48500.000000	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.000000	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.000000	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	45333.333333	2020-11-25	Bangalore	NaN

2) Axis = 1, csv.interpolate(method='linear',axis=1),

Its will fill by assuming linear and along with row and its not default.

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6048\2217450782.py in ?()
----> 1 fl.interpolate(method='linear', axis=1)

c:\Users\DELL\anaconda3\Lib\site-packages\pandas\core\generic.py in ?(self, method, axis, limit, inplace, limit_direction, limit_a
   8469
   8470         limit_direction = missing.infer_limit_direction(limit_direction, method)
   8471
   8472         if obj.ndim == 2 and np.all(obj.dtypes == object):
-> 8473             raise TypeError(
   8474                 "Cannot interpolate with all object-dtype columns "
   8475                 "in the DataFrame. Try setting at least one "
   8476                 "column to a numeric dtype."

TypeError: Cannot interpolate with all object-dtype columns in the DataFrame. Try setting at least one column to a numeric dtype.
```

- It will show error cause all the entity should have **same datatype**.

1) Interpolate with limit -

- It will fill only limited no of entity as we mention
- It will work only with consecutive blank or null entity which is in series.
- **var.interpolate(limit = value)**

```
In [16]: var.interpolate(limit=2)
```

Out[16]:

	NO.	SYMBOL	SECURITY	TODAY - VOLUME	AVG. VOLUME	CHANGE
0	1	ELECTHERM	Electrotherm (India) Limited	473907.0	9457.0	50.110003
1	2	NBIFIN	N. B. I. Industrial Finance Company Limited	5754.0	154.0	37.202586
2	3	HINDCON	Hindcon Chemicals Limited	304759.0	9248.0	32.953450
3	4	GKWLIMITED	GKW Limited	21375.0	711.0	30.049203
4	5	21STCENMGM	21st Century Management Services Limited	110375.0	4533.5	31.049203
5	6	NaN	Alphageo (India) Limited	104147.0	8356.0	27.852613
6	7	UNIENTER	Uniphos Enterprises Limited	97919.0	8357.0	21.528002
7	8	JAGSNPHARM	Jagsonpal Pharmaceuticals Limited	NaN	48066.0	17.801420
8	9	SKMEGGPROD	SKM Egg Products Export (India) Limited	NaN	87775.0	18.801420
9	10	SWELECTES	Swelect Energy Systems Limited	NaN	87776.0	16.938733
10	11	NAHARPOLY	Nahar Poly Films Limited	NaN	38574.0	15.555297

It has also direction like limit with direction…

a) **Forward direction** - it will fill only above null value

- **csv.interpolate(limit_direction= “forward”, limit=value)**

```
In [17]: var.interpolate(limit_direction="forward", limit=2)
```

Out[17]:

	NO.	SYMBOL	SECURITY	TODAY - VOLUME	AVG. VOLUME	CHANGE
0	1	ELECTHERM	Electrotherm (India) Limited	473907.0	9457.0	50.110003
1	2	NBIFIN	N. B. I. Industrial Finance Company Limited	5754.0	154.0	37.202586
2	3	HINDCON	Hindcon Chemicals Limited	304759.0	9248.0	32.953450
3	4	GKWLIMITED	GKW Limited	21375.0	711.0	30.049203
4	5	21STCENMGM	21st Century Management Services Limited	110375.0	4533.5	31.049203
5	6	NaN	Alphageo (India) Limited	104147.0	8356.0	27.852613
6	7	UNIENTER	Uniphos Enterprises Limited	97919.0	8357.0	21.528002
7	8	JAGSNPHARM	Jagsonpal Pharmaceuticals Limited	NaN	48066.0	17.801420
8	9	SKMEGGPROD	SKM Egg Products Export (India) Limited	NaN	87775.0	18.801420
9	10	SWELECTES	Swelect Energy Systems Limited	NaN	87776.0	16.938733
10	11	NAHARPOLY	Nahar Poly Films Limited	NaN	38574.0	15.555297
11	12	PODDARHOUS	Poddar Housing and Development Limited	NaN	3012.0	15.283833

b) **Backword direction** - it will fill only below null value

- **csv.interpolate(limit_direction= “backword”, limit=value)**

```
In [19]: var.interpolate(limit_direction="backward", limit=2)
```

Out[19]:

	NO.	SYMBOL	SECURITY	TODAY - VOLUME	AVG. VOLUME	CHANGE
0	1	ELECTHERM	Electrotherm (India) Limited	473907.0	9457.0	50.110003
1	2	NBIFIN	N. B. I. Industrial Finance Company Limited	5754.0	154.0	37.202586
2	3	HINDCON	Hindcon Chemicals Limited	304759.0	9248.0	32.953450
3	4	GKWLIMITED	GKW Limited	21375.0	711.0	30.049203
4	5	21STCENMGM	21st Century Management Services Limited	110375.0	4533.5	31.049203
5	6	NaN	Alphageo (India) Limited	NaN	8356.0	27.852613
6	7	UNIENTER	Uniphos Enterprises Limited	NaN	8357.0	21.528002
7	8	JAGSNPHARM	Jagsonpal Pharmaceuticals Limited	NaN	48066.0	17.801420
8	9	SKMEGGPROD	SKM Egg Products Export (India) Limited	NaN	87775.0	18.801420
9	10	SWELECTES	Swelect Energy Systems Limited	NaN	87776.0	16.938733
10	11	NAHARPOLY	Nahar Poly Films Limited	73007.0	38574.0	15.555297
11	12	PODDARHOUS	Poddar Housing and Development Limited	66779.0	3012.0	15.283833
12	13	LINC	Linc Limited	60551.0	4076.0	14.853673
13	14	LINC123	Linc Limited123	60550.0	4075.0	15.853673

c) **<u>Both direction</u> -** it will fill from above and below both null values

- **csv.interpolate(limit_direction= “both”, limit=value)**

```
In [20]: var.interpolate(limit_direction="both", limit=2)
```

Out[20]:

	NO.	SYMBOL	SECURITY	TODAY - VOLUME	AVG. VOLUME	CHANGE
0	1	ELECTHERM	Electrotherm (India) Limited	473907.0	9457.0	50.110003
1	2	NBIFIN	N. B. I. Industrial Finance Company Limited	5754.0	154.0	37.202586
2	3	HINDCON	Hindcon Chemicals Limited	304759.0	9248.0	32.953450
3	4	GKWLIMITED	GKW Limited	21375.0	711.0	30.049203
4	5	21STCENMGM	21st Century Management Services Limited	110375.0	4533.5	31.049203
5	6	NaN	Alphageo (India) Limited	104147.0	8356.0	27.852613
6	7	UNIENTER	Uniphos Enterprises Limited	97919.0	8357.0	21.528002
7	8	JAGSNPHARM	Jagsonpal Pharmaceuticals Limited	NaN	48066.0	17.801420
8	9	SKMEGGPROD	SKM Egg Products Export (India) Limited	NaN	87775.0	18.801420
9	10	SWELECTES	Swelect Energy Systems Limited	NaN	87776.0	16.938733
10	11	NAHARPOLY	Nahar Poly Films Limited	73007.0	38574.0	15.555297

2) Interpolate with limit_area -

- It tell like should we use interpolate or not…

a) It fill null value normally - csv.interpolate(limit_area = “inside”)

- Check salary column

```
fl.interpolate(limit_area="inside")
```

✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_6048\2701774836.py:1: FutureWarning:
  fl.interpolate(limit_area="inside")
```

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.000000	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	48500.000000	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.000000	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.000000	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	45333.333333	2020-11-25	Bangalore	NaN

b) It don’t fill null value - csv.interpolate(limit_area = “outside”)

- Check salary column

```
fl.interpolate(limit_area="outside")
```
✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_6048\1230911464.py:1: FutureWarn
  fl.interpolate(limit_area="outside")
```

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.0	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	NaN	2020-11-25	Bangalore	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3) Interpolate with inplace

- It make change the null value permanently (if inplace = Treu) or just a copy (if inplace = False).
- But not in original csv file, it make just a copy for there in code or idle.

a) Permanently value.

(csv.interpolate(limit_area="inside",inplace=True) or any other function.

```
fl.interpolate(limit_area="inside",inplace=True)
fl
```
✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_6048\433620942.py:1: FutureWarning: D
  fl.interpolate(limit_area="inside",inplace=True)
```

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.000000	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	48500.000000	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.000000	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.000000	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	45333.333333	2020-11-25	Bangalore	NaN

And when we later check —

```
fl
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.000000	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	48500.000000	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.000000	NaN	Pune	NaN

b) Temporary value.
(csv.interpolate(limit_area="inside",inplace=False) or any other function.

```
cd.interpolate(limit_area="inside",inplace=False)
```
✓ 0.0s

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_6048\2979310623.py:1: FutureWarning:
  cd.interpolate(limit_area="inside",inplace=False)
```

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.000000	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	48500.000000	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.000000	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.000000	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	45333.333333	2020-11-25	Bangalore	NaN
5	105.5	NaN	NaN	42666.666667	NaN	NaN	NaN
6	106.0	Aman	Sales	40000.000000	2021-06-18	Delhi	NaN

And … when we check later…

```
cd
```
✓ 0.0s

	EmployeeID	Name	Department	Salary	JoinDate	City	Bonus
0	101.0	Rahul	IT	45000.0	2022-01-15	Delhi	NaN
1	102.0	Ankit	HR	NaN	2021-08-10	Mumbai	NaN
2	103.0	Neha	Finance	52000.0	NaN	Pune	NaN
3	104.0	Rohan	IT	48000.0	2023-03-01	NaN	NaN
4	105.0	Priya	Marketing	NaN	2020-11-25	Bangalore	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Merging and conct.

Merging and Concatenate

1) Merging- merge two or more series or dataframe as joints.

1) Normal merging

- But need a same series in both dataframe which use for merge along.
- **pd.merge(s1,s2,on="A")**

Row data -

```
s1 =
pd.DataFrame({'A':[1,2,3,4,6],'B':[10,20,30,40,60]})
s2 =
pd.DataFrame({'A':[1,2,3,4,6],'c':[11,22,33,44,66]})
```

```
[12]  pd.merge(s1,s2,on="A")
      ✓ 0.0s
```

	A	B	c
0	1	10	11
1	2	20	22
2	3	30	33
3	4	40	44
4	6	60	66

And if same series like(A) if there is few elements is different then…

Like, row data-

```
s1 =
pd.DataFrame({'A':[1,2,3,4,6],'B':[10,20,30,40,60]})
s2 =
pd.DataFrame({'A':[1,2,3,5,7],'c':[11,22,33,44,66]})
S1
```

```
pd.merge(s1,s2,on="A")
✓ 0.0s
```

	A	B	c
0	1	10	11
1	2	20	22
2	3	30	33

2) Merge with how … (pd.merge(var1,var2,how="__")

- **How** have many parameters like - "inner","left","right"

a) Inner or by default

- It work normal as merge work…

```
pd.merge(s1,s2,how='inner')
```
✓ 0.0s

	A	B	c
0	1	10	11
1	2	20	22
2	3	30	33

b) Left

- It merge according first variable like (s1) cause it is leftest one. But according same series

```
pd.merge(s1,s2,how='left')
```
✓ 0.0s

	A	B	c
0	1	10	11.0
1	2	20	22.0
2	3	30	33.0
3	4	40	NaN
4	6	60	NaN

c) Right

- It merge according second variable like (s2) cause it is rightest one. But according same series

```
pd.merge(s1,s2,how="right")
```
✓ 0.0s

	A	B	c
0	1	10.0	11
1	2	20.0	22
2	3	30.0	33
3	5	NaN	44
4	7	NaN	66

d) Outer

- It merge both variable by having null value… like (full join)

```
pd.merge(s1,s2,how="outer")
```
✓ 0.0s

	A	B	c
0	1	10.0	11.0
1	2	20.0	22.0
2	3	30.0	33.0
3	4	40.0	NaN
4	5	NaN	44.0
5	6	60.0	NaN
6	7	NaN	66.0

e) Indicator

- It shows indicator with any joins, and tell like both joints or not

```
pd.merge(s1,s2,how="outer",indicator=True)
```
✓ 0.0s

	A	B	c	_merge
0	1	10.0	11.0	both
1	2	20.0	22.0	both
2	3	30.0	33.0	both
3	4	40.0	NaN	left_only
4	5	NaN	44.0	right_only
5	6	60.0	NaN	left_only
6	7	NaN	66.0	right_only

3) Merge if both variable have same header.

row data-

```
s3 = pd.DataFrame({'A':[1,2,3,4],'B':[10,20,30,40]})
s4 = pd.DataFrame({'A':[1,2,3,5],'B':[11,22,33,44]})
```

- It shows nothing if we just merge it normally, like this-

```
pd.merge(s3,s4)
```
✓ 0.0s

A	B

1) that's why for merge these both as a table we use,

```
pd.merge(s3,s4,left_index=True,right_index=True)
```

```
pd.merge(s3,s4,left_index=True,right_index=True)
```

✓ 0.0s

	A_x	B_x	A_y	B_y
0	1	10	1	11
1	2	20	2	22
2	3	30	3	33
3	4	40	5	44

2) And if i need to change header name so,

```
pd.merge(s3,s4,left_index=True,right_index=True,
suffixes=("VAR1","VAR2"))
```

```
pd.merge(s3,s4,left_index=True,right_index=True,suffixes=("VAR1","VAR2"))
```

✓ 0.0s

	AVAR1	BVAR1	AVAR2	BVAR2
0	1	10	1	11
1	2	20	2	22
2	3	30	3	33
3	4	40	5	44

2) Concat- merge or join two or more series or dataframe, In both direction just below or just along.

1) Normal Concat

Row data (A,B hedr) -

```
d3 = pd.DataFrame({'A':[1,2,3,4],'b':[10,20,30,40]})
d4 = pd.DataFrame({'A':[1,2,3,5],'b':[11,22,33,44]})
```

```
pd.concat([d3,d4])
```

0.0s

	A	b
0	1	10
1	2	20
2	3	30
3	4	40
0	1	11
1	2	22
2	3	33
3	5	44

And if, Row data (A,B,C - 3 diff hdr) is -

```
d1 = pd.DataFrame({'A':[1,2,3,4],'b':[10,20,30,40]})
d2 = pd.DataFrame({'A':[1,2,3,5],'c':[11,22,33,44]})
```

```
pd.concat([d1,d2],axis=0)
```

0.0s

	A	b	c
0	1	10.0	NaN
1	2	20.0	NaN
2	3	30.0	NaN
3	4	40.0	NaN
0	1	NaN	11.0
1	2	NaN	22.0
2	3	NaN	33.0
3	5	NaN	44.0

axis = 0 is by default

And if, we change Axis = 1, then

```
pd.concat([d3,d4],axis=1)
```

0.0s

	A	b	A	b
0	1	10	1	11
1	2	20	2	22
2	3	30	3	33
3	4	40	5	44

2) Concat with joins.

pd.concat([d5,d6],axis=1 or 0,join='outer' or 'inner')

Row data

```
d5 = pd.DataFrame({'A':[1,2,3,4],'b':[10,20,30,40]})
d6 = pd.DataFame({'A':[1,2],'b':[11,22]})
```

1) **Join (outer)** - it will show as full join

```
pd.concat([d5,d6],axis=1,join='outer')
```

✓ 0.0s

	A	b	A	b
0	1	10	1.0	11.0
1	2	20	2.0	22.0
2	3	30	NaN	NaN
3	4	40	NaN	NaN

2) **Join (inner)** - it will show as inner join like which is only matching.

```
pd.concat([d5,d6],axis=1,join='inner')
```

✓ 0.0s

	A	b	A	b
0	1	10	1	11
1	2	20	2	22

3) Concat with keys.

- It shows just heading of variables.

```
pd.concat([d5,d6],axis=1,keys=["d1","d2"])
```

✓ 0.0s

	d1		d2	
	A	b	A	b
0	1	10	1.0	11.0
1	2	20	2.0	22.0
2	3	30	NaN	NaN
3	4	40	NaN	NaN

```
pd.concat([d5,d6],axis=0,keys=["d1","d2"])
```
✓ 0.0s

		A	b
d1	0	1	10
	1	2	20
	2	3	30
	3	4	40
d2	0	1	11
	1	2	22

4) If variable have no same no. of series or columns

```
d7 = pd.DataFrame({'A':[1,2,3,4]})
d8 = pd.DataFrame({'A':[1,2,3,5],'b':[11,22,33,44]})
```
[53] ✓ 0.0s

```
pd.concat([d7,d8])
```
[55] ✓ 0.0s

	A	b
0	1	NaN
1	2	NaN
2	3	NaN
3	4	NaN
0	1	11.0
1	2	22.0
2	3	33.0
3	5	44.0

```
d7 = pd.DataFrame({'c':[1,2,3,4]})
d8 = pd.DataFrame({'A':[1,2,3,5],'b':[11,22,33,44]})
```
[57] ✓ 0.0s

```
pd.concat([d7,d8])
```
[58] ✓ 0.0s

	c	A	b
0	1.0	NaN	NaN
1	2.0	NaN	NaN
2	3.0	NaN	NaN
3	4.0	NaN	NaN
0	NaN	1.0	11.0
1	NaN	2.0	22.0
2	NaN	3.0	33.0
3	NaN	5.0	44.0

join and append

Join and Append

1) Join- join two or more series or dataframe as joints or merge.

1) Normal join

- But need a same series in both dataframe which use for merge along.

```
j1 = pd.DataFrame({'A':[1,2,3,4],'b':[10,20,30,40]})
j2 = pd.DataFrame({'D':[1,2,3,5],'c':[11,22,33,44]})
```

✓ 0.0s

```
j1.join(j2)
```

✓ 0.0s

	A	b	D	c
0	1	10	1	11
1	2	20	2	22
2	3	30	3	33
3	4	40	5	44

If entity is less in variable 2 -

```
j1 = pd.DataFrame({'A':[1,2,3,4],'b':[10,20,30,40]})
j2 = pd.DataFrame({'D':[1,2],'c':[11,22]})
```

✓ 0.0s

```
j1.join(j2)
```

✓ 0.0s

	A	b	D	c
0	1	10	1.0	11.0
1	2	20	2.0	22.0
2	3	30	NaN	NaN
3	4	40	NaN	NaN

- **If we change index no of any variable**

```
j1 = pd.DataFrame({'A':[1,2,3,4],'b':[10,20,30,40]},index=["a","b","c","d"])
j2 = pd.DataFrame({'D':[1,2],'c':[11,22]})
```
✓ 0.0s

```
j1.join(j2)
```
✓ 0.0s

	A	b	D	c
a	1	10	NaN	NaN
b	2	20	NaN	NaN
c	3	30	NaN	NaN
d	4	40	NaN	NaN

- So need to also write index no in 2nd var same as no of entity and also same as (“a”,”b”) not (“g”,”f”)

```
j1 = pd.DataFrame({'A':[1,2,3,4],'b':[10,20,30,40]},index=["a","b","c","d"])
j2 = pd.DataFrame({'D':[1,2],'c':[11,22]},index=["c","a"])
```
[67] ✓ 0.0s

```
j1.join(j2)
```
[68] ✓ 0.0s

	A	b	D	c
a	1	10	2.0	22.0
b	2	20	NaN	NaN
c	3	30	1.0	11.0
d	4	40	NaN	NaN

```
j1 = pd.DataFrame({'A':[1,2,3,4],'b':[10,20,30,40]},index=["a","b","c","d"])
j2 = pd.DataFrame({'D':[1,2],'c':[11,22]},index=["a","b"])
j1.join(j2)
```
[69] ✓ 0.0s

	A	b	D	c
a	1	10	1.0	11.0
b	2	20	2.0	22.0
c	3	30	NaN	NaN
d	4	40	NaN	NaN

- And if we change variable front or backword so output will also change accordingly

```
j2.join(j1)
```
✓ 0.0s

	D	c	A	b
a	1	11	1	10
b	2	22	2	20

- And we can also check all type of output by having same positions like having j2 first and then j1 = j2.join(j1) by how = “inner”, “outer”, “left”, “right”

1) **how = “inner”** - will behave as inner join

```
j2.join(j1,how="inner")
```
0.0s

	D	c	A	b
a	1	11	1	10
b	2	22	2	20

2) **how = “outer”** - will behave as full join

```
j2.join(j1,how='outer')
```
0.0s

	D	c	A	b
a	1.0	11.0	1	10
b	2.0	22.0	2	20
c	NaN	NaN	3	30
d	NaN	NaN	4	40

3) **how = “left”** - will behave as left join, so according j2 because it is in left

```
j2.join(j1,how='left')
```
0.0s

	D	c	A	b
a	1	11	1	10
b	2	22	2	20

4) **how = “right”** - will behave as right join, so according j1 because it is in left

```
j2.join(j1,how='right')
```
0.0s

	D	c	A	b
a	1.0	11.0	1	10
b	2.0	22.0	2	20
c	NaN	NaN	3	30
d	NaN	NaN	4	40

2) And if we have same name of header in both variables, so we can use of suffix.

like , Row data -

```
j1 =
pd.DataFrame({'A':[1,2,3,4],'B':[10,20,30,40]},index=["a","b","c","d"])
j2 =
pd.DataFrame({'D':[1,2],'B':[11,22]},index=["a","b"])
```

It will show error -

```
columns overlap but no suffix specified: Index(['B'], dtype='object')
```

So, Solutions is -

var1.join(var2,how="any",lsuffix='any_dg or no.')

```
j1 = pd.DataFrame({'A':[1,2,3,4],'B':[10,20,30,40]},index=["a","b","c","d"])
j2 = pd.DataFrame({'D':[1,2],'B':[11,22]},index=["a","b"])
j1.join(j2,how="outer",lsuffix='new')
```

✓ 0.0s

	A	Bnew	D	B
a	1	10	1.0	11.0
b	2	20	2.0	22.0
c	3	30	NaN	NaN
d	4	40	NaN	NaN

```
j1 = pd.DataFrame({'A':[1,2,3,4],'B':[10,20,30,40]},index=["a","b","c","d"])
j2 = pd.DataFrame({'D':[1,2],'B':[11,22]},index=["a","b"])
j1.join(j2,how="outer",lsuffix='new',rsuffix='0')
```

✓ 0.0s

	A	Bnew	D	B0
a	1	10	1.0	11.0
b	2	20	2.0	22.0
c	3	30	NaN	NaN
d	4	40	NaN	NaN

```
j1 = pd.DataFrame({'A':[1,2,3,4],'B':[10,20,30,40]},index=["a","b","c","d"])
j2 = pd.DataFrame({'D':[1,2],'B':[11,22]},index=["a","b"])
j1.join(j2,how="outer",lsuffix='0',rsuffix='0')
```

✓ 0.0s

	A	B0	D	B0
a	1	10	1.0	11.0
b	2	20	2.0	22.0
c	3	30	NaN	NaN
d	4	40	NaN	NaN

even we can keep same name on both suffix.

2) Append- join two or more series or dataframe as joints or merge.

There is major difference b/w append and join like as we need rsuffix and lsuffix when we add two variable if there have same no of header, but in append it doesn't impact.

The `append()` function is no longer a part of the pandas DataFrame API in recent versions (2.0 and later). It was deprecated in version 1.4.0 and completely removed to encourage the use of more efficient and explicit methods

Timeline (clear and simple)

- **2021** → Deprecated (pandas **1.4**)
- **2022** → Still present, but with warnings (pandas **1.5**)
- **2023** → **Completely removed** (pandas **2.0** release)

So from **2023 onward**, `append()` **does not exist** in pandas.

✅ Correct replacement (recommended)

Use `pd.concat()`

```
import pandas as pd

df = pd.concat([df, new_df], ignore_index=True)
```

Append a single row

```
new_row = pd.DataFrame([{"Name": "Rahul", "Dept": "IT"}])
df = pd.concat([df, new_row], ignore_index=True)
```

❌ Why `append()` was removed

- Very **slow** in loops
- Encouraged **bad performance patterns**
- `concat()` is faster and clearer

If you see `append()` working somewhere

That code is:

- Old pandas (<1.4), or
- Using a **Series** (Series.append is also removed now)

group by

Group by

- **It just conclude whole table or group….**

Row data -

GROUP BY

Generate + Code + Markdown

```
gvar = pd.DataFrame({"Name":["a","b","a","c","b","b","a","d","a","c","d","d"],
                     "sub1":[12,42,32,39,34,56,30,23,72,32,93,82],
                     "sub2":[46,27,86,76,56,86,45,35,86,34,87,45]})

gvar
```

✓ 0.0s

	Name	sub1	sub2
0	a	12	46
1	b	42	27
2	a	32	86
3	c	39	76
4	b	34	56
5	b	56	86
6	a	30	45
7	d	23	35
8	a	72	86
9	c	32	34
10	d	93	87
11	d	82	45

So, group by

```
var_new = gvar.groupby("Name")
```

✓ 0.0s

```
for x,y in var_new:
    print("")
    print(x)

    print(y)
```

✓ 0.0s

```
a
  Name  sub1  sub2
0    a    12    46
2    a    32    86
6    a    30    45
8    a    72    86
```

```
b
  Name  sub1  sub2
1    b    42    27
4    b    34    56
5    b    56    86

c
  Name  sub1  sub2
3    c    39    76
9    c    32    34

d
   Name  sub1  sub2
7     d    23    35
10    d    93    87
11    d    82    45
```

Here x tells - (a,b,c,d) and y gives all values

- **And if you want group by of only one name Var.get_group("name")**

```
var_new.get_group("a")
```

✓ 0.0s

	Name	sub1	sub2
0	a	12	46
2	a	32	86
6	a	30	45
8	a	72	86

```
var_new.get_group('d')
```

✓ 0.0s

	Name	sub1	sub2
7	d	23	35
10	d	93	87
11	d	82	45

- **And also we can get descriptions Like - min, max,**
  - a) **Min - var.min()**

```
var_new.min()
```
✓ 0.0s

Name	sub1	sub2
a	12	45
b	34	27
c	32	34
d	23	35

  - b) **Max - var.max()**

```
var_new.max()
```
✓ 0.0s

Name	sub1	sub2
a	72	86
b	56	86
c	39	76
d	93	87

  - c) **Mean - var.mean()**

```
var_new.mean()
```
✓ 0.0s

Name	sub1	sub2
a	36.5	65.750000
b	44.0	56.333333
c	35.5	55.000000
d	66.0	55.666667

  - d) **And so on…**

- **And if you want to make all group by data as lists.**

```
li = list(var_new)
li
```

0.0s

```
[('a',
    Name  sub1  sub2
  0    a    12    46
  2    a    32    86
  6    a    30    45
  8    a    72    86),
 ('b',
    Name  sub1  sub2
  1    b    42    27
  4    b    34    56
  5    b    56    86),
 ('c',
    Name  sub1  sub2
  3    c    39    76
  9    c    32    34),
 ('d',
     Name  sub1  sub2
  7     d    23    35
  10    d    93    87
  11    d    82    45)]
```

reshape pd file

Reshape Pandas File
Pivot Table and Melt

1) Melt - it reshape dataframe of series.

- Or if you want to change all tabular form data in one column
- pd.melt(var)

Row data -

```
var = pd.DataFrame({"days":[1,2,3,4,5,6],"eng":[43,64,63,46,94,34],"math":[94,29,49,64,30,54]})
var
```

0.0s

	days	eng	math
0	1	43	94
1	2	64	29
2	3	63	49
3	4	46	64
4	5	94	30
5	6	34	54

1) pd.melt(var)

```
pd.melt(var)
```

[5] 0.0s

	variable	value
0	days	1
1	days	2
2	days	3
3	days	4
4	days	5
5	days	6
6	eng	43
7	eng	64
8	eng	63
9	eng	46
10	eng	94
11	eng	34
12	math	94
13	math	29
14	math	49
15	math	64
16	math	30
17	math	54

2) We can also assign it by different id…

- pd.metl(csv,id_vars =[‘column_name’]

```
pd.melt(var,id_vars=['days'])
```

✓ 0.0s

	days	variable	value
0	1	eng	43
1	2	eng	64
2	3	eng	63
3	4	eng	46
4	5	eng	94
5	6	eng	34
6	1	math	94
7	2	math	29
8	3	math	49
9	4	math	64
10	5	math	30
11	6	math	54

3) And we can also change Variable name and Value name which is header

- pd.melt(csv,var_name = ‘any’,value_name = ‘any’)

```
pd.melt(var,id_vars='days',var_name='Subject',value_name='Marks')
```

✓ 0.0s

	days	Subject	Marks
0	1	eng	43
1	2	eng	64
2	3	eng	63
3	4	eng	46
4	5	eng	94
5	6	eng	34
6	1	math	94
7	2	math	29
8	3	math	49
9	4	math	64
10	5	math	30
11	6	math	54

2) Pivot- it conclude dataframe of series.

- It just conclude whole tabular meaning of table.

1) Normal Pivot

```
var1.pivot(index='days',columns='Student')
```

Row data -

```
var1 =
pd.DataFrame({"days":[1,2,3,4,5,6],"Student":['a','b',
'b','c','a','c'],"eng":[43,64,63,46,94,34],"math":[94,
29,49,64,30,54]})
var1
```

```
var1 = pd.DataFrame({"days":[1,2,3,4,5,6],"Student":[
var1
```

0.0s

	days	Student	eng	math
0	1	a	43	94
1	2	b	64	29
2	3	b	63	49
3	4	c	46	64
4	5	a	94	30
5	6	c	34	54

Now,

```
var1.pivot(index='days',columns='Student')
```

0.0s

	eng			math		
Student	a	b	c	a	b	c
days						
1	43.0	NaN	NaN	94.0	NaN	NaN
2	NaN	64.0	NaN	NaN	29.0	NaN
3	NaN	63.0	NaN	NaN	49.0	NaN
4	NaN	NaN	46.0	NaN	NaN	64.0
5	94.0	NaN	NaN	30.0	NaN	NaN
6	NaN	NaN	34.0	NaN	NaN	54.0

Other example if there is only (a and b) as student

```
var2 = pd.DataFrame({"days":[1,2,3,4,5,6],"Student":['a','b','b','a','a','b'],"e
var2
```

0.0s

	days	Student	eng	math
0	1	a	43	94
1	2	b	64	29
2	3	b	63	49
3	4	a	46	64
4	5	a	94	30
5	6	b	34	54

```
var2.pivot(index='days',columns='Student')
```

0.0s

	eng		math	
Student	a	b	a	b
days				
1	43.0	NaN	94.0	NaN
2	NaN	64.0	NaN	29.0
3	NaN	63.0	NaN	49.0
4	46.0	NaN	64.0	NaN
5	94.0	NaN	30.0	NaN
6	NaN	34.0	NaN	54.0

2) We can get also details of specific columns by pivot

- **`var2.pivot(index='any_col',columns='any_col', values='any_col which you want only')`**

```
var2.pivot(index='days',columns='Student',values='math')
```

0.0s

Student	a	b
days		
1	94.0	NaN
2	NaN	29.0
3	NaN	49.0
4	64.0	NaN
5	30.0	NaN
6	NaN	54.0

3) Now, arithmetic functions by pivot (like - means,sum,etc)
   - aggfunc (which means arthm.. fun)
   - `var3.pivot_table(index='days',columns='Student', aggfunc='mean')`

   - 1) aggfunc = 'mean'

```
var3.pivot_table(index='days',columns='Student',
                 aggfunc='mean')
```

✓ 0.0s

	eng		math	
Student	a	b	a	b
days				
1	54.0	NaN	59.0	NaN
2	NaN	64.0	NaN	42.0

   - 2) aggfunc = 'sum'

```
var3.pivot_table(index='days',columns='Student',
                 aggfunc='sum')
```

✓ 0.0s

	eng		math	
Student	a	b	a	b
days				
1	216.0	NaN	236.0	NaN
2	NaN	128.0	NaN	84.0

4) Pivot_table with Margin
   - It just a total or other arithmetic by according whole column or whole row in all terms.

For sum -

```
var3.pivot_table(index='days',columns='Student',
                 aggfunc='sum',margins=True)
```

✓ 0.0s

	eng			math		
Student	**a**	**b**	**All**	**a**	**b**	**All**
days						
1	216.0	NaN	216	236.0	NaN	236
2	NaN	128.0	128	NaN	84.0	84
All	216.0	128.0	344	236.0	84.0	320

For mean -

```
var3.pivot_table(index='days',columns='Student',
                 aggfunc='mean',margins=True)
```

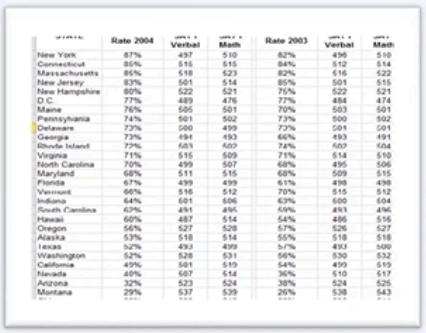
✓ 0.0s

	eng			math		
Student	**a**	**b**	**All**	**a**	**b**	**All**
days						
1	54.0	NaN	54.000000	59.0	NaN	59.000000
2	NaN	64.0	64.000000	NaN	42.0	42.000000
All	54.0	64.0	57.333333	59.0	42.0	53.333333

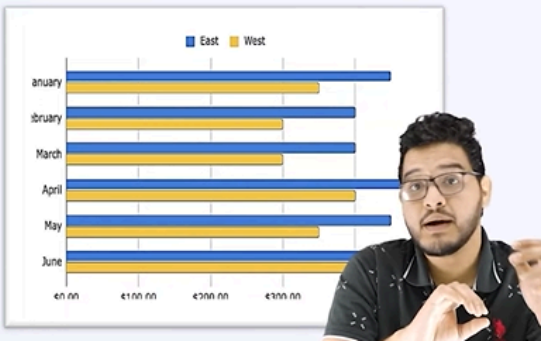
Matplotlib

Why Data Visualization ?

- Data visualization is a quick, easy way to convey concepts in a universal manner.



	Rate 2004	Verbal	Math	Rate 2003	Verbal	Math
New York	87%	497	510	82%	496	510
Connecticut	85%	515	515	84%	512	514
Massachusetts	85%	518	523	82%	516	522
New Jersey	83%	501	514	85%	501	515
New Hampshire	80%	522	521	75%	522	521
D.C.	77%	489	476	77%	484	474
Maine	76%	505	501	70%	503	501
Pennsylvania	74%	501	502	73%	500	502
Delaware	73%	500	499	73%	501	501
Georgia	73%	494	493	66%	493	491
Rhode Island	73%	501	502	74%	502	504
Virginia	71%	515	509	71%	514	510
North Carolina	70%	499	507	68%	495	506
Maryland	68%	511	515	68%	509	515
Florida	67%	499	499	61%	498	498
Vermont	66%	516	512	70%	515	512
Indiana	64%	501	506	63%	500	504
South Carolina	62%	491	495	59%	493	496
Hawaii	60%	487	514	54%	486	516
Oregon	56%	527	528	57%	526	527
Alaska	53%	518	514	55%	518	518
Texas	52%	493	499	51%	493	500
Washington	52%	528	531	56%	530	532
California	49%	501	519	54%	499	519
Nevada	49%	507	514	36%	510	517
Arizona	32%	523	524	38%	524	525
Montana	29%	537	539	26%	538	543



What is Data Visualization

- Data visualization is the graphical representation of information and data.
- visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

Type of Data Visualization in Python

Plotting Libraries:

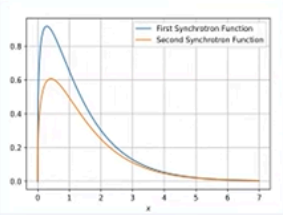
- **Matplotlib:** low level, provides lots of freedom
- **Pandas Visualization:** easy to use interface, built on Matplotlib
- **Seaborn:** high-level interface, great default styles
- **ggplot:** based on R's ggplot2, uses Grammar of Graphics
- **Plotly:** can create interactive plots

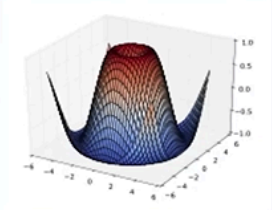
What you will learn?

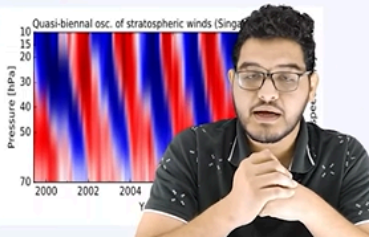
- what is matplotlib ?
- Matplotlib Graphs
- Installation of matplotlib
- Importing Matplot

What is Matplotlib ?

- Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.
- Matplotlib is a python library used for Data Visualization.
- Matplotlib is 2D and 3D plotting python library.
- It was introduced by john hunter in the year 2002.







- Linear Plot
- Scatter Plot
- Bar Plot
- Stem Plot
- Step Plot
- Hist Plot
- Box Plot
- Pie Plot
- Fill_Between Plot

install & import

Install and Import

1) Install

Two ways -

1) **By cmd…**
   Pip install matplotlib
2) **And, By Python Terminal…**
   Pip install matplotlib

2) Import

import matplotlib.pyplot as plt

3) Check version

Three ways -

1) **By cmd…**
   a) **Pip list**
   b) **Pip show matplotlib**

2) **By python terminal…**
   a) **Pip list**
   b) **Pip show matplotlib**

3) **By importing library**

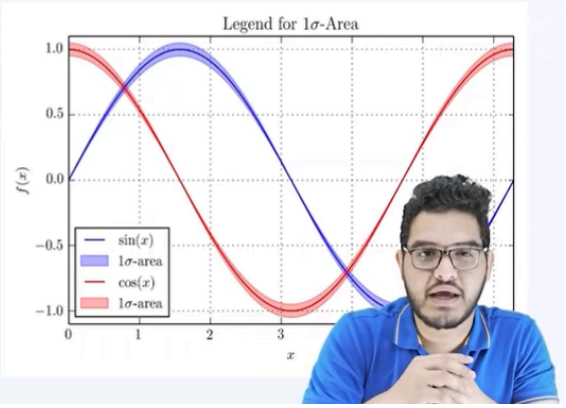
```
Import matplotlib
print(matplotlib.__version__)
```

Line and fill_between plot

Line graph and Fill Between Plot

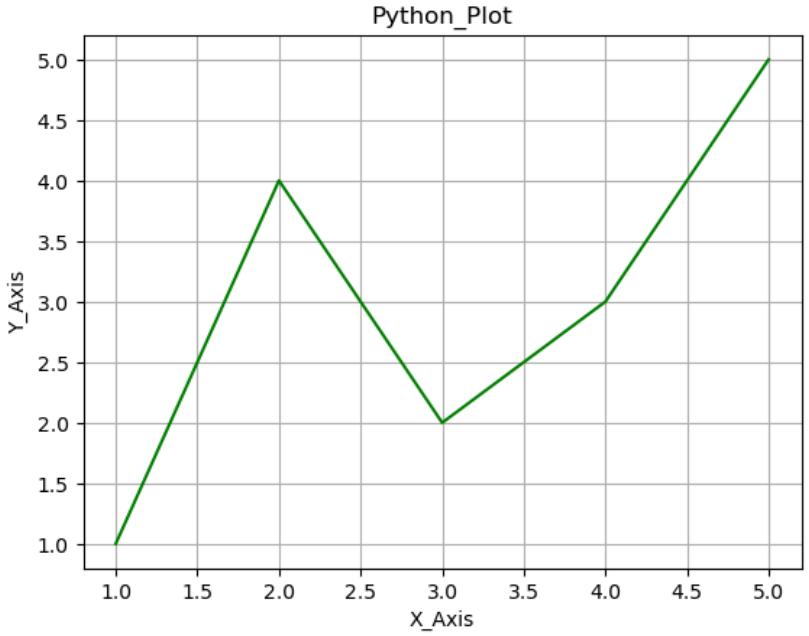
Matplotlib Fill_Between Plot

```
import matplotlib.pyplot as plt
x = [ ]
y = [ ]
plt. plot(x,y)
plt.fill_between( )
plt.show()
```



1) Line graph

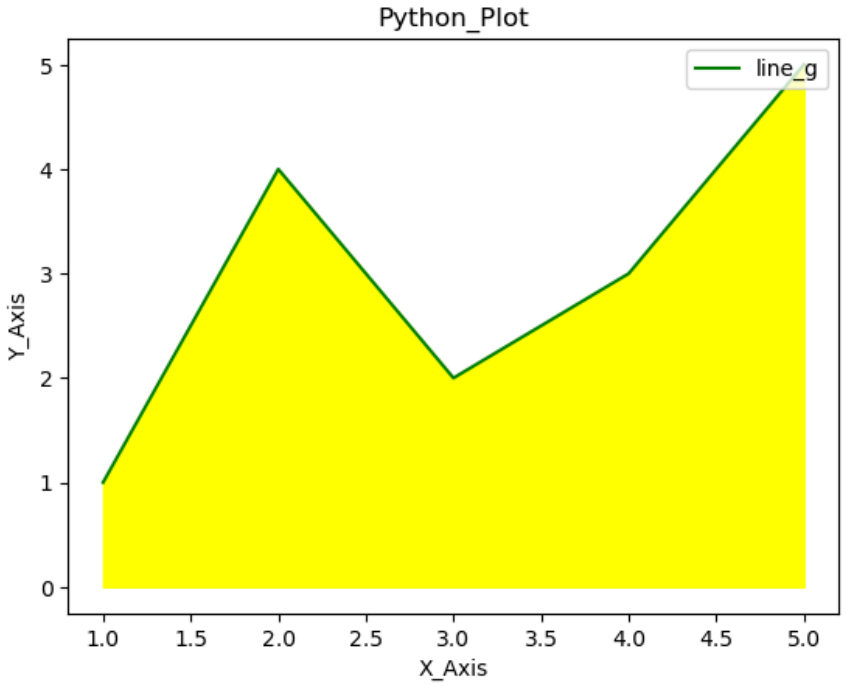
```
x = [1,2,3,4,5]
y = [1,4,2,3,5]

plt.plot(x,y,color="g")
plt.title("Python_Plot")
plt.xlabel("X_Axis")
plt.ylabel("Y_Axis")
plt.grid()
plt.show()
```



2) Fill_between

a) Normal fill _between

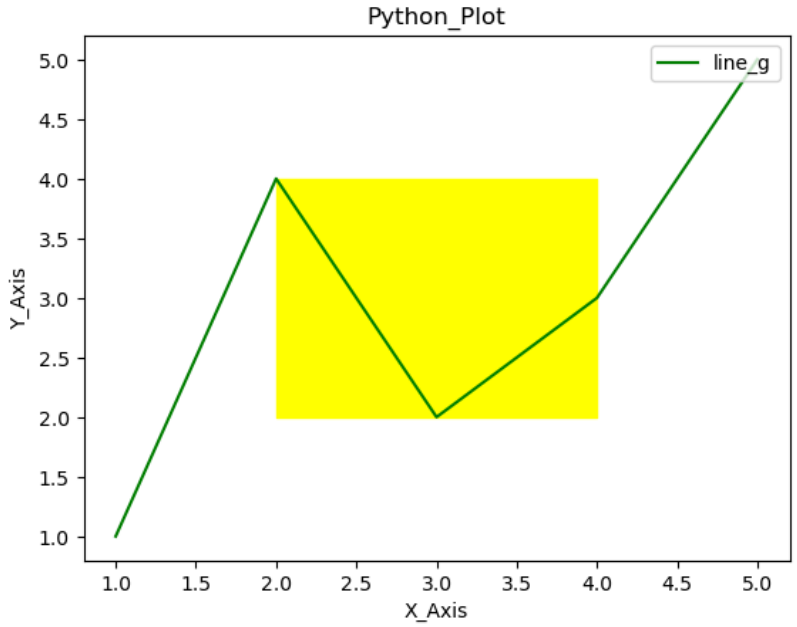
```
x = [1,2,3,4,5]
y = [1,4,2,3,5]

plt.plot(x,y,color="g",label="line_g")

plt.fill_between(x,y,color="yellow")

plt.legend(loc=1)
plt.title("Python_Plot")
plt.xlabel("X_Axis")
plt.ylabel("Y_Axis")
plt.show()
```



b) Fill_between - between specific coordinates of x and y

```
x = [1,2,3,4,5]
y = [1,4,2,3,5]

plt.plot(x,y,color="g",label="line_g")

plt.fill_between(x=[2,4],y1=2,y2=4,color="yellow")

plt.legend(loc=1)
```

```
plt.title("Python_Plot")
plt.xlabel("X_Axis")
plt.ylabel("Y_Axis")
plt.show()
```



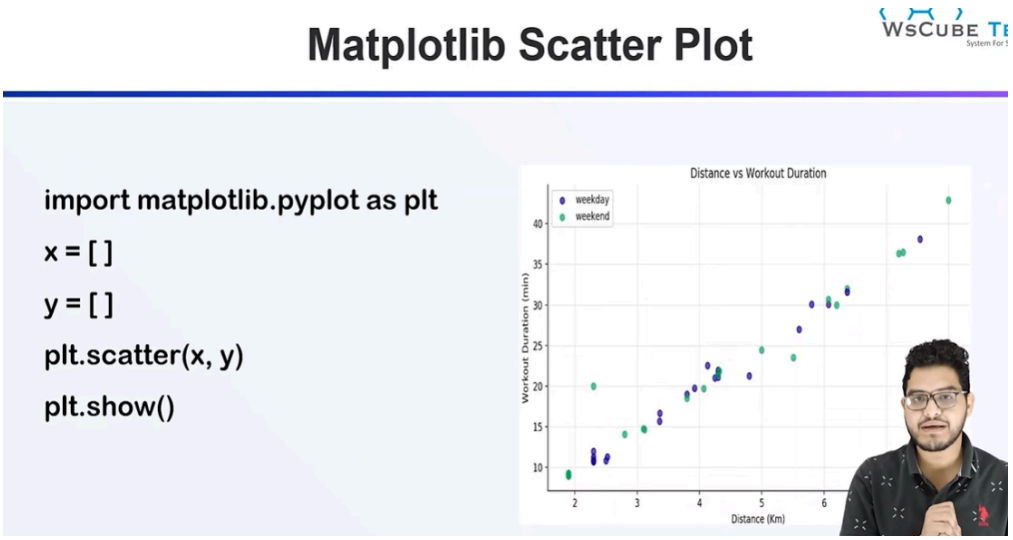
c) Fill_between - by only specific coordinates of x

```
import numpy as np

x = np.array([1,2,3,4,5])
y = np.array([1,4,2,3,5])

plt.plot(x,y,color="g",label="line_g")

plt.fill_between(x,y,color="yellow",where=(x>=2)&(x<=4
),alpha=0.3)

plt.legend(loc=1)
plt.title("Python_Plot")
plt.xlabel("X_Axis")
plt.ylabel("Y_Axis")
plt.show()
```

Python_Plot
line_g
Y_Axis
X_Axis
0
1
2
3
4
5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Scatter graph plot

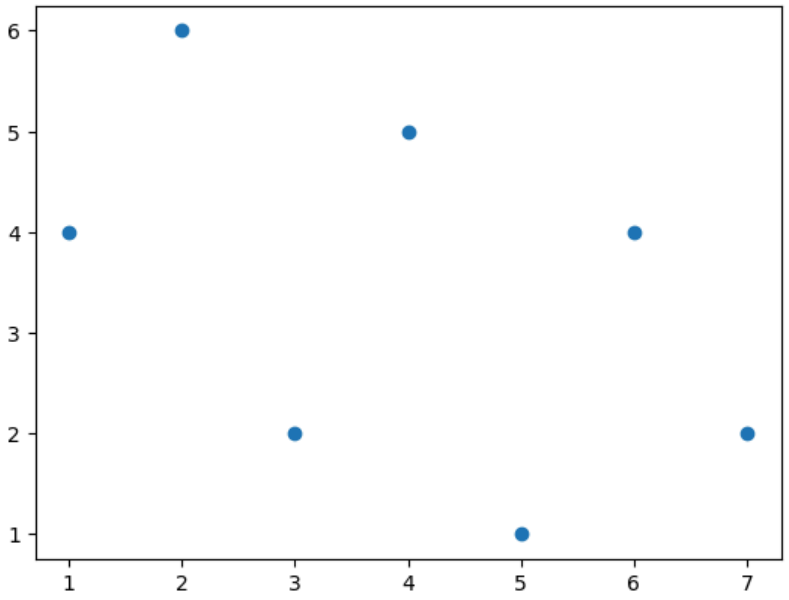
Scatter Plot



1) Normal

```
import matplotlib.pyplot as plt

x = [1,2,3,4,5,6,7]
y = [4,6,2,5,1,4,2]
plt.scatter(x,y)
plt.show()
```

Output -

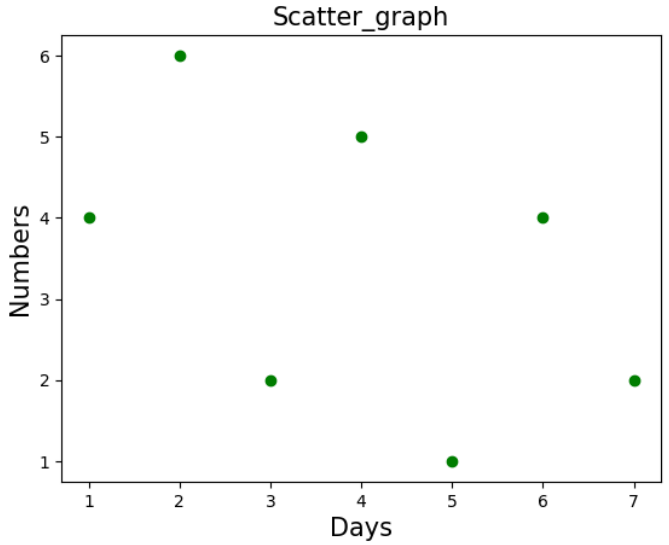


2) Modified

a) Normal Modified - like (colour,title,and legends)

```
x = [1,2,3,4,5,6,7]
y = [4,6,2,5,1,4,2]
plt.scatter(x,y,color="g")
plt.title("Scatter_graph",fontsize = 15)
plt.xlabel("Days",fontsize = 15)
plt.ylabel("Numbers",fontsize= 15)
plt.show()
```

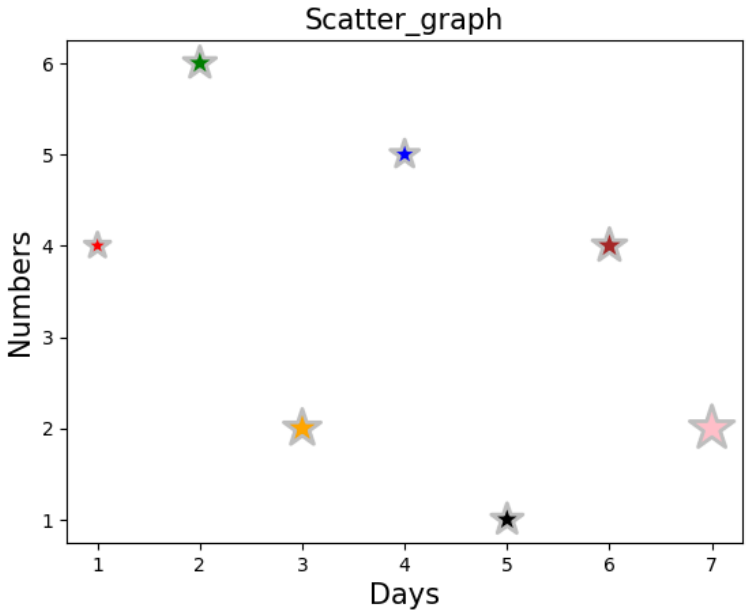
Output -



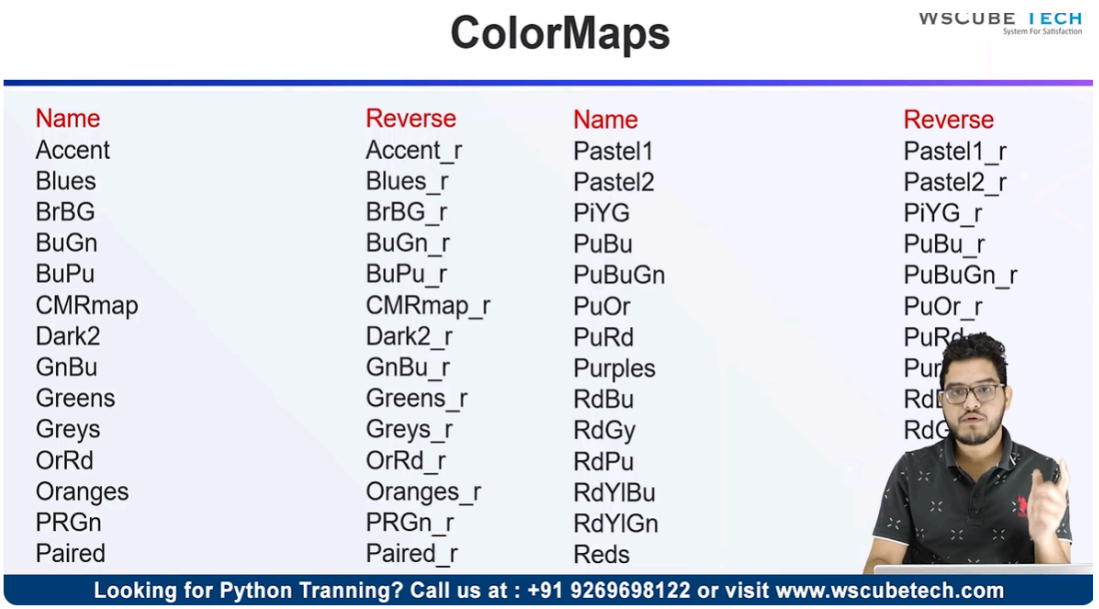
b) Complex modified -

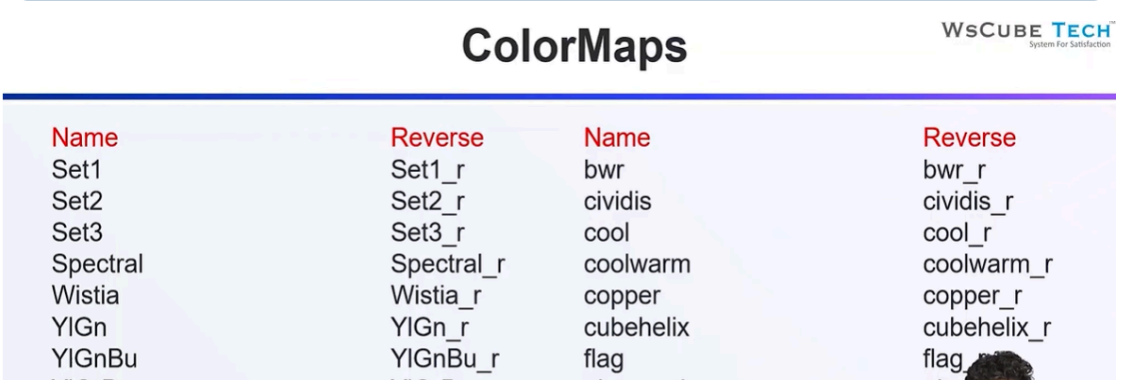
```
x = [1,2,3,4,5,6,7]
y = [4,6,2,5,1,4,2]
c = ["r","g","orange","blue","black","brown","pink"]

s = [200,320,420,250,300,350,600]
plt.scatter(x,y,color=c,sizes =
s,marker="*",edgecolors="silver",linewidths=2)

plt.title("Scatter_graph",fontsize = 15)
plt.xlabel("Days",fontsize = 15)
plt.ylabel("Numbers",fontsize= 15)
plt.show()
```

Output -



c) Complex modified 2 -

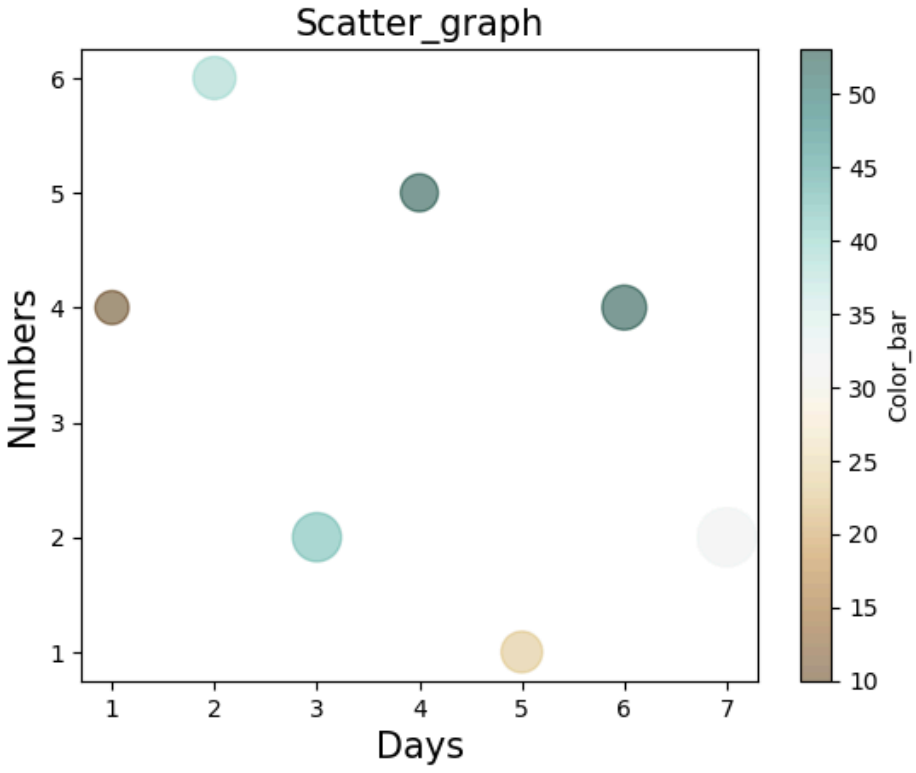




YlGnBu	YlGnBu_r	flag	flag_
YlOrBr	YlOrBr_r	gist_earth	gist_ _r
YlOrRd	YlOrRd_r	gist_gray	gist_ _r
afmhot	afmhot_r	gist_heat	gist_ _r
autumn	autumn_r	gist_ncar	
binary	binary_r	gist_rainbow	
bone	bone_r	gist_stern	
brg	brg_r	gist_yarg	

```
x = [1,2,3,4,5,6,7]
y = [4,6,2,5,1,4,2]
c = [10,39,42,53,23,53,32]
s = [200,320,420,250,300,350,600]

plt.scatter(x,y,c=c,sizes = s,cmap="BrBG",alpha=0.5)
t = plt.colorbar()
t.set_label("Color_bar")

plt.title("Scatter_graph",fontsize = 15)
plt.xlabel("Days",fontsize = 15)
plt.ylabel("Numbers",fontsize= 15)
plt.show()
```

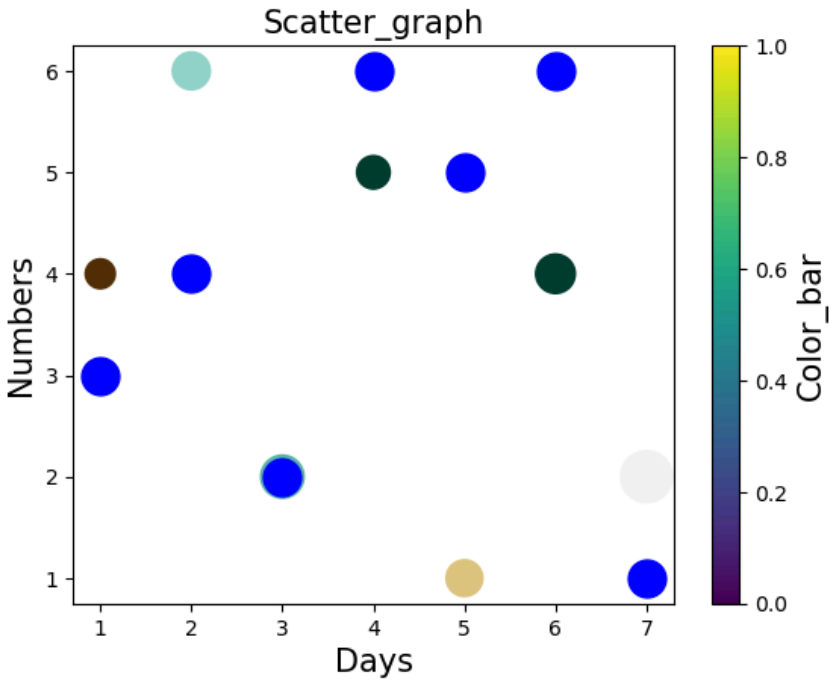
Output -



d) Complex modified 3 - two data set

```
x = [1,2,3,4,5,6,7]
y = [4,6,2,5,1,4,2]
z = [3,4,2,6,5,6,1]

c = [10,39,42,53,23,53,32]
s = [200,320,420,250,300,350,600]

plt.scatter(x,y,c=c,sizes = s,cmap="BrBG")
plt.scatter(x,z,color="blue",s=320)
t = plt.colorbar()
t.set_label("Color_bar",fontsize=15)

plt.title("Scatter_graph",fontsize = 15)
plt.xlabel("Days",fontsize = 15)
plt.ylabel("Numbers",fontsize= 15)
plt.show()
```

Output -



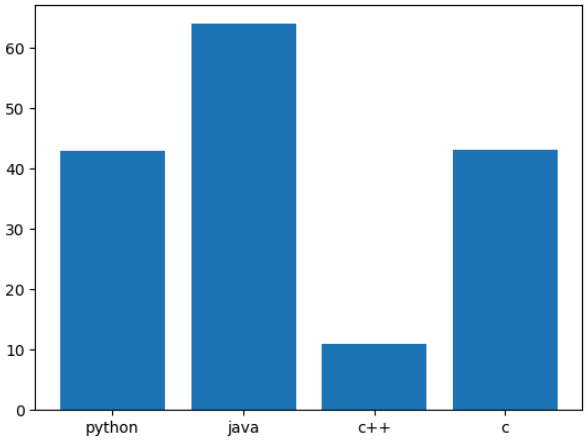
Bar graph plot

Bar Plot

1) Normal

```
x = ["python","java","c++","c"]
y = [43,64,11,43.2]
plt.bar(x,y)
plt.show()
```

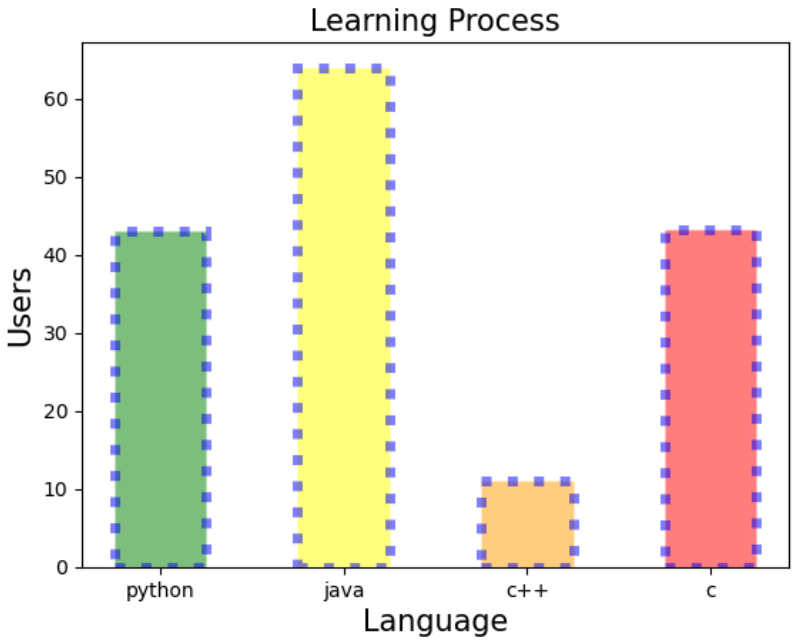
Output -



2) Modified bar chart -

```
x = ["python","java","c++","c"]
y = [43,64,11,43.2]
plt.xlabel("Language", fontsize=15) #-- for x label
plt.ylabel("Users",fontsize = 15)     #-- for y label
plt.title("Learning Process",fontsize = 15) #-- for title
c = ["green","yellow","orange","red"]
plt.bar(x,y,width=0.5,color=c,edgecolor='blue',linewidth=5,l
inestyle=":",alpha=0.5)
# -- linewidht, linestyle , alpha means = dim the colour
(b/w - 1 to 0)
plt.show()
```

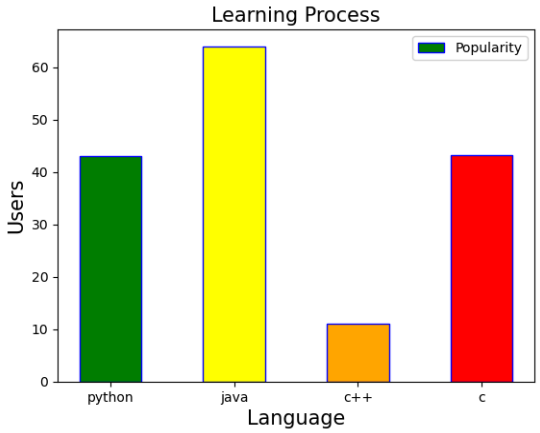
Output -



- **For getting label in bar graph.**

```
x = ["python","java","c++","c"]
y = [43,64,11,43.2]
plt.xlabel("Language", fontsize=15) #-- for x label
plt.ylabel("Users",fontsize = 15)     #-- for y label
plt.title("Learning Process",fontsize = 15) #-- for title
c = ["green","yellow","orange","red"]
plt.bar(x,y,width=0.5,color=c,edgecolor='blue',label="Popula
rity")
plt.legend()
# -- for label -- legend is mandatory
plt.show()
```

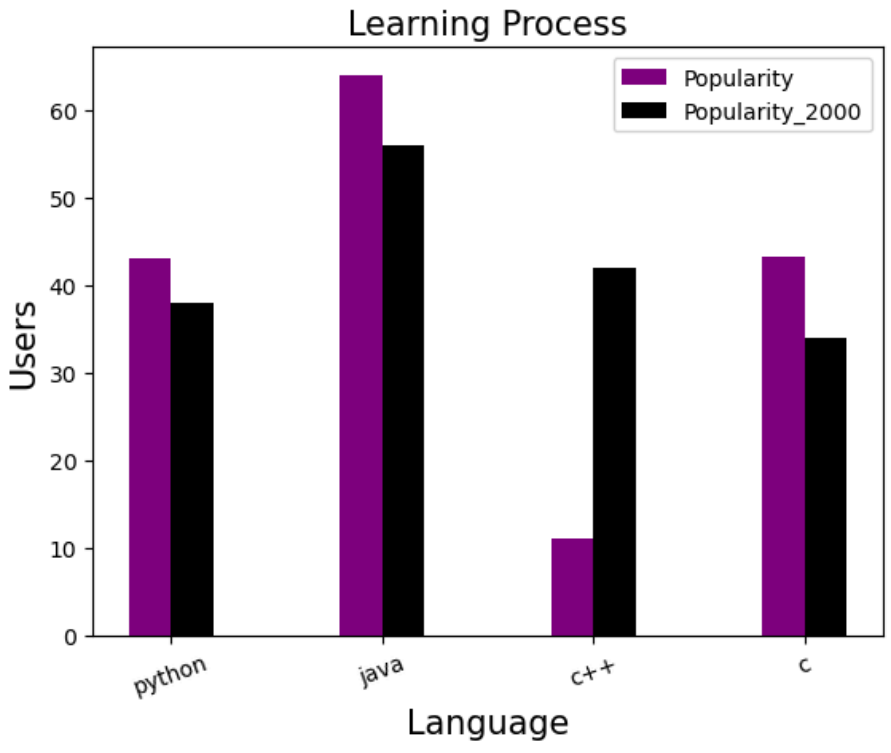
Output -



- **For getting two graph or labels in one graph**

```
import numpy as np

x = ["python","java","c++","c"]
y = [43,64,11,43.2]
z = [38,56,42,34]

width=0.2
p = np.arange(len(x))
p1 = [j+width for j in p]

plt.xlabel("Language", fontsize=15)
plt.ylabel("Users",fontsize = 15)
plt.title("Learning Process",fontsize = 15)

plt.bar(p,y,width,color="purple",label="Popularity")
plt.bar(p1,z,width,color="black",label="Popularity_2000")

plt.xticks(p+width/2,x,rotation = 20)
plt.legend()
plt.show()
```

Output -



Why NumPy is used in your code

p = np.arange(len(x))

This creates numeric positions for the bars:

[0, 1, 2, 3]

Then:

p1 = [j + width for j in p]

shifts the second set of bars slightly to the right so the bars appear side-by-side instead of overlapping.

In short:

- **Matplotlib needs numeric x positions to place bars**
- **NumPy makes generating and manipulating those positions easy and clean**
- **`np.arange()` is faster and more readable than manual lists**

Can you get the same output WITHOUT NumPy?

✅ Yes, absolutely.

You just need to create the positions manually using Python lists.

Same output without NumPy

```
import matplotlib.pyplot as plt

x = ["python","java","c++","c"]
y = [43,64,11,43.2]
z = [38,56,42,34]

width = 0.2

# manual positions
p = [0, 1, 2, 3]
```

```
p1 = [i + width for i in p]

plt.xlabel("Language", fontsize=15)
plt.ylabel("Users", fontsize=15)
plt.title("Learning Process", fontsize=15)

plt.bar(p, y, width, color="purple", label="Popularity")
plt.bar(p1, z, width, color="black", label="Popularity_2000")

plt.xticks([i + width/2 for i in p], x, rotation=20)
plt.legend()
plt.show()
```

✔️ **Output will be identical**

Then why do most people use NumPy?

1. Less manual work
np.arange(len(x))

instead of

[0, 1, 2, 3]

2. Works automatically for any data size

If you add more languages:

x.append("go")

NumPy adapts automatically — no need to change the positions list.

3. Cleaner mathematical operations
p + width/2

works directly with NumPy arrays, but not with Python lists.

When NumPy is recommended

✔ Large datasets
✔ Dynamic charts
✔ Mathematical transformations
✔ Professional / production plots

Summary

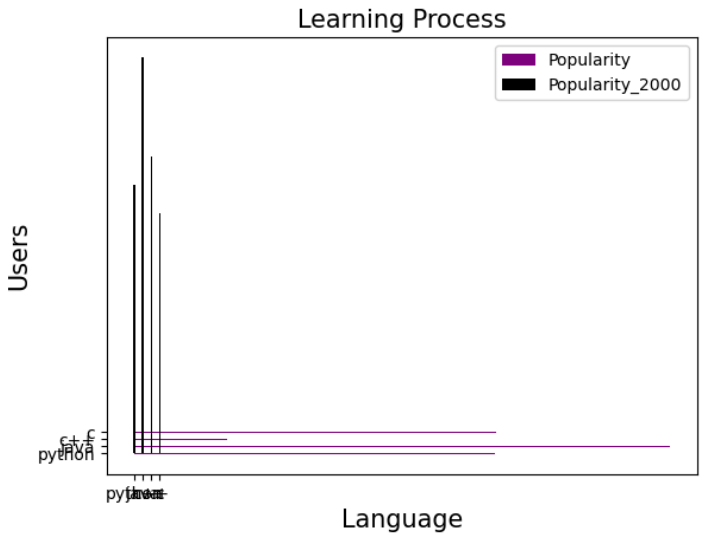
- ❌ NumPy is not mandatory
- ✅ You *can* do it without NumPy
- ⭐ NumPy makes plotting cleaner, scalable, and professional

3) Horizontal bar graph

Just add “h” in bar, like = “barh”

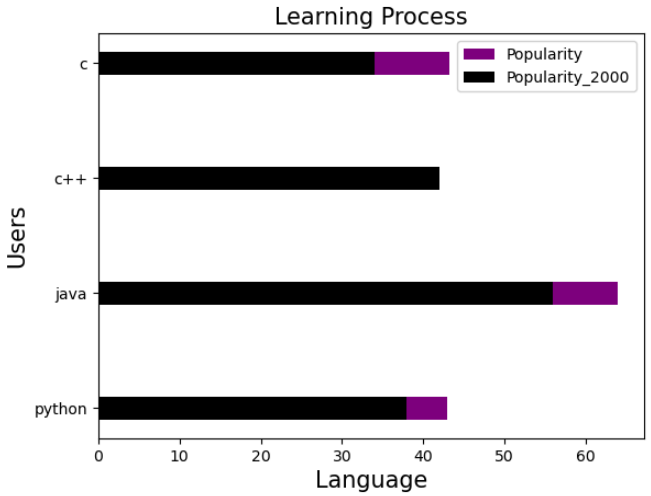
Eg1 -

```
x = ["python","java","c++","c"]
y = [43,64,11,43.2]
z = [38,56,42,34]

plt.xlabel("Language", fontsize=15)
plt.ylabel("Users",fontsize = 15)
plt.title("Learning Process",fontsize = 15)

plt.barh(x,y,width,color="purple",label="Popularity")
plt.bar(x,z,width,color="black",label="Popularity_2000")

plt.legend()
plt.show()

Output -
```



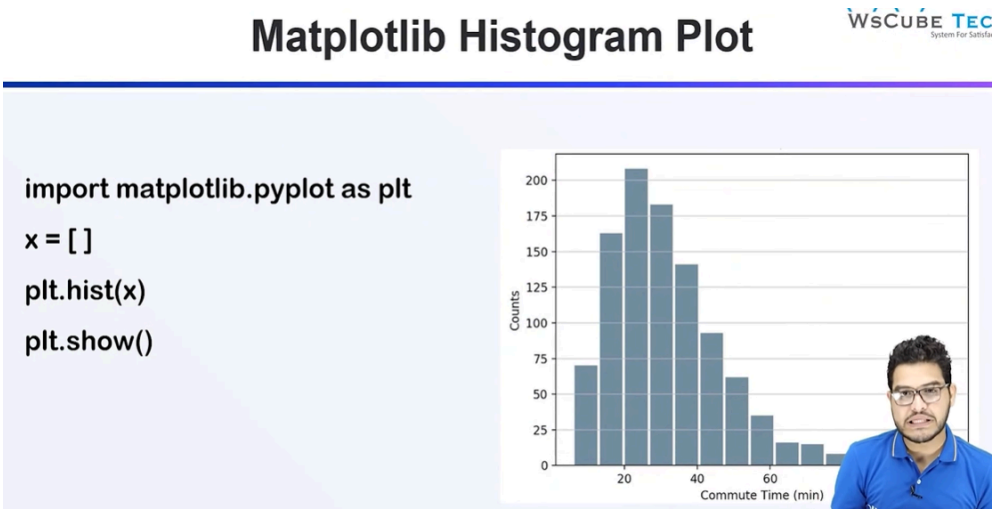
Eg2 -

```
x = ["python","java","c++","c"]
y = [43,64,11,43.2]
z = [38,56,42,34]

plt.xlabel("Language", fontsize=15)
plt.ylabel("Users",fontsize = 15)
plt.title("Learning Process",fontsize = 15)

plt.barh(x,y,width,color="purple",label="Popularity")
plt.barh(x,z,width,color="black",label="Popularity_2000")

plt.legend()
plt.show()
```



Histogram plot

Histogram Plot



1) Normal

HISTOGRAM PLOT

```
[3]  import matplotlib.pyplot as plt
     ✓ 0.3s
```

for getting some fake data we will use random functions

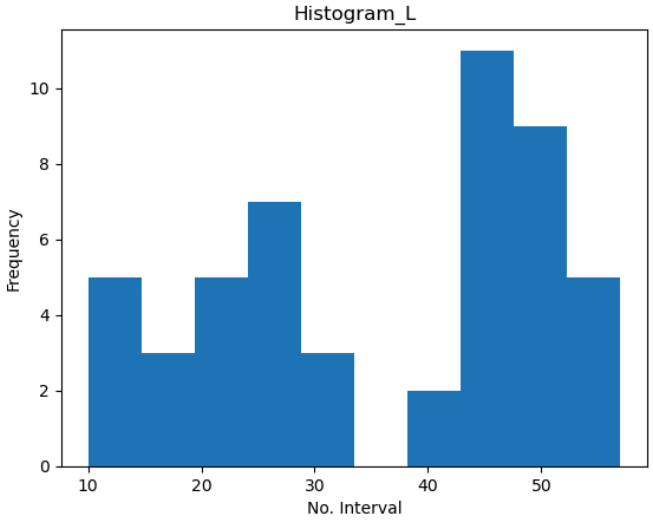
```
[7]  import numpy as np
     import random

     v = np.random.randint(10,60,(50))
     v
     ✓ 0.0s
```

```
array([46, 46, 56, 22, 51, 23, 47, 43, 19, 44, 14, 46, 31, 50, 20, 27, 51,
       10, 55, 10, 11, 50, 50, 11, 52, 50, 44, 42, 44, 28, 20, 41, 27, 57,
       28, 26, 46, 31, 54, 47, 27, 43, 48, 16, 52, 22, 56, 30, 18, 25])
```

Now, we have fix data

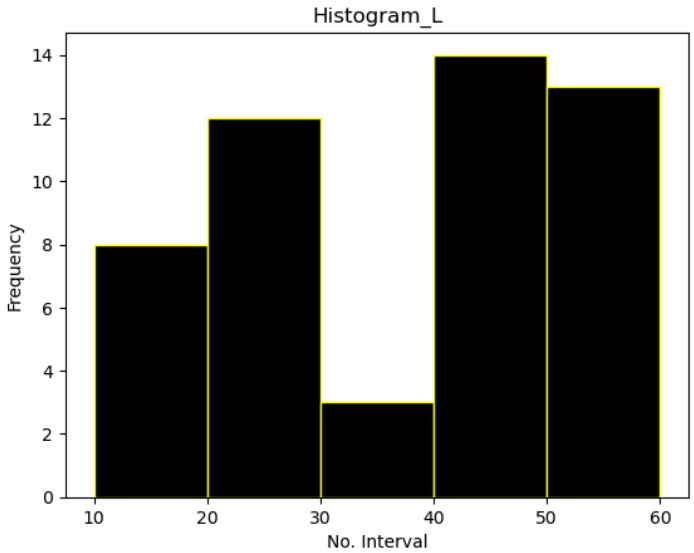
```
x = [46, 46, 56, 22, 51, 23, 47, 43, 19, 44, 14, 46, 31, 50, 20, 27, 51,
     10, 55, 10, 11, 50, 50, 11, 52, 50, 44, 42, 44, 28, 20, 41, 27, 57,
     28, 26, 46, 31, 54, 47, 27, 43, 48, 16, 52, 22, 56, 30, 18, 25]

l = [10,20,30,40,50,60]
```

```
plt.hist(x)

plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")
plt.show()
```



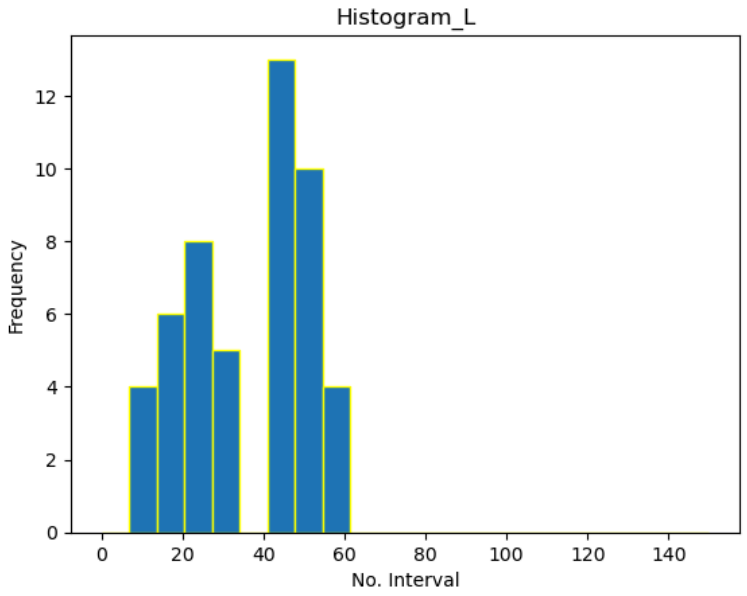
2) But above is not a full set or valid histogram,
So for set - use bins…
But you can't limit intervals like - (0,200) here.

```
l = [10,20,30,40,50,60]

plt.hist(x,color='black',bins=l,edgecolor='yellow')

plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")

plt.show()
```



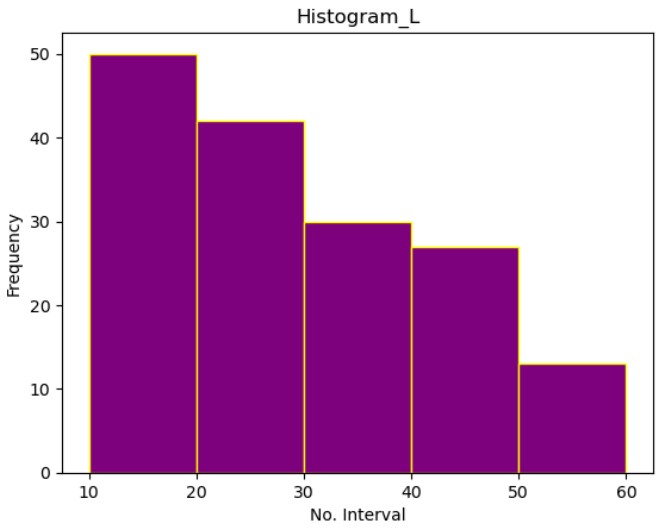
And you want set but don't want to use bin - so you can use "auto", and you can also limit intervals like - (0,200) But - you can't use colour, here

```
l = [10,20,30,40,50,60]

plt.hist(x,"auto",(0,150),edgecolor='yellow')

plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")

plt.show()
```



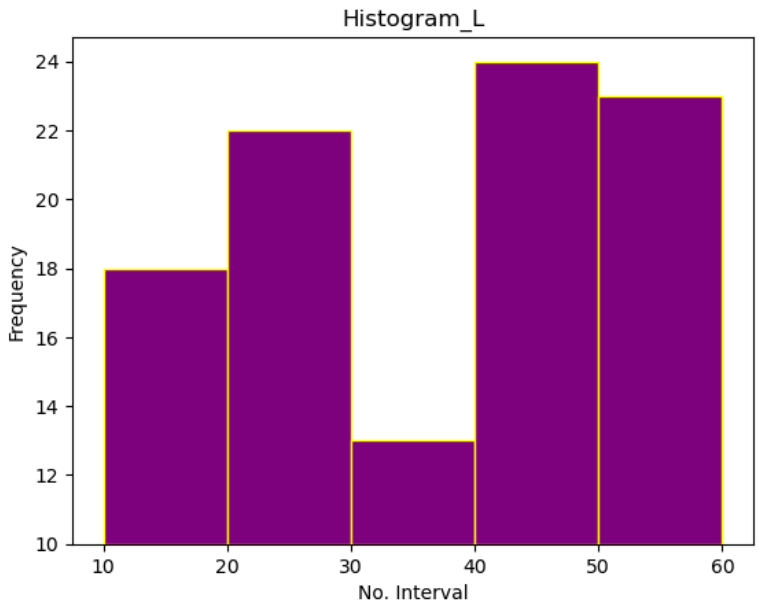
3) Cumulative - its reverse histogram frequency (-1 and 1)

```
l = [10,20,30,40,50,60]
plt.hist(x,color="purple",bins=l,edgecolor='yellow',cumulati
ve=-1)

plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")
plt.show()
```



4) Bottom and align

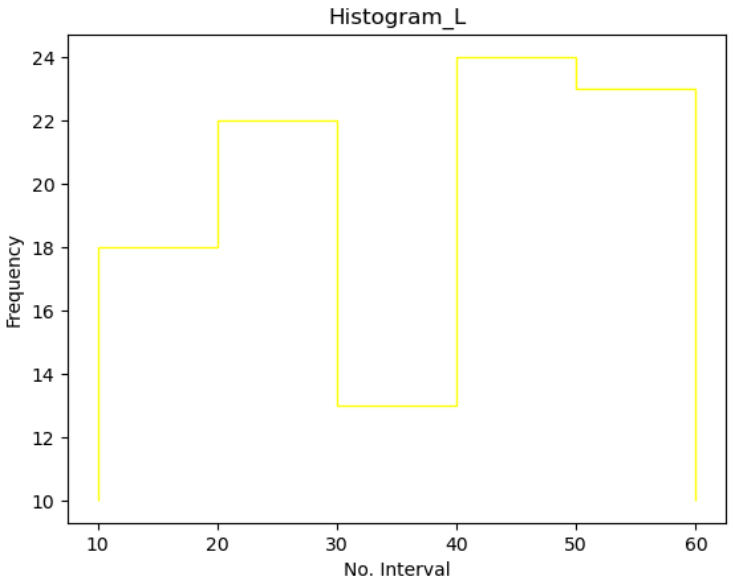
```
l = [10,20,30,40,50,60]
plt.hist(x,color="purple",bins=l,edgecolor='yellow',bottom=1
0,align='mid')

plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")

plt.show()
```



5) histtype

- **“Step”**

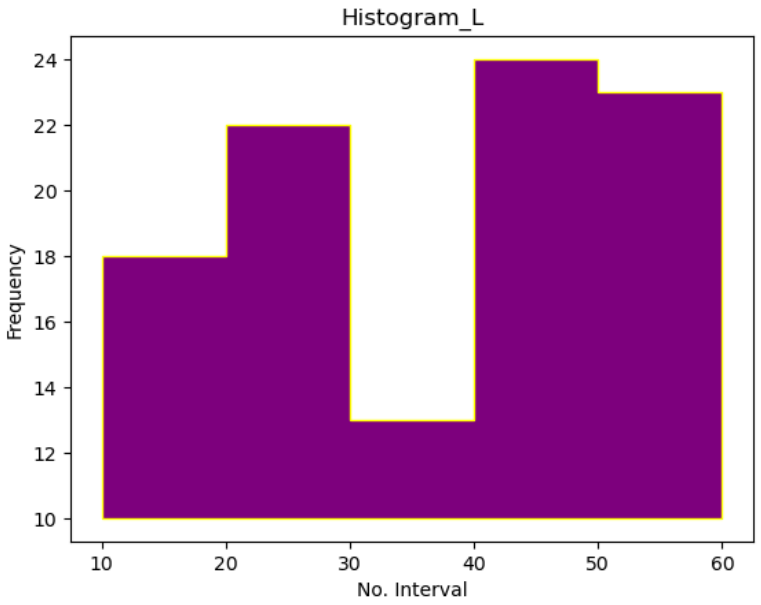
```
l = [10,20,30,40,50,60]
plt.hist(x,color="purple",bins=l,edgecolor='yellow',bottom=1
0,align='mid',histtype='step')
plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")
```



- **stepfilled**

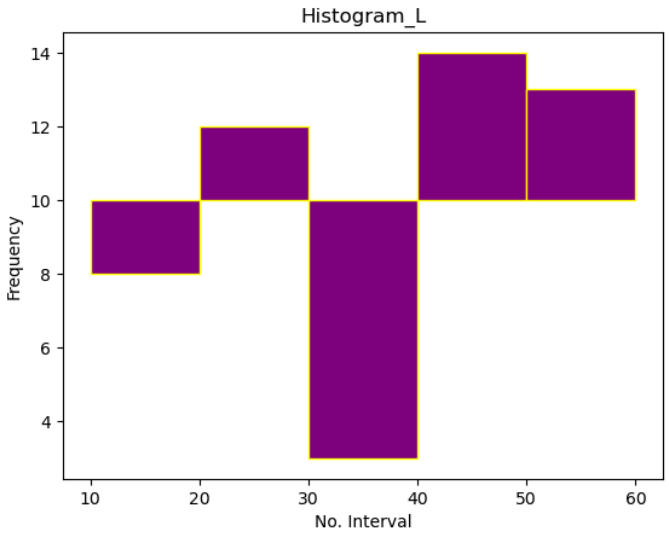
```
plt.hist(x,color="purple",bins=l,edgecolor='yellow',bottom=1
0,align='mid',histtype="stepfilled")
```

Rest all same…



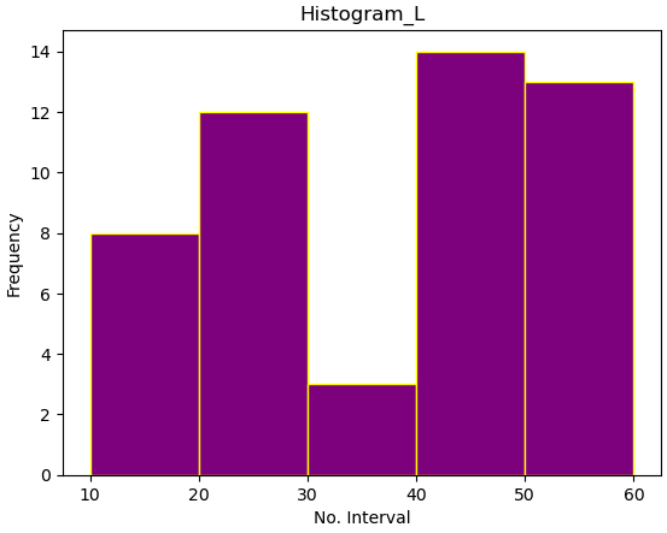
- **barstacked**

```
plt.hist(x,color="purple",bins=l,edgecolor='yellow',bottom=1
0,align='mid',histtype="barstacked")
```



if bottom = 10

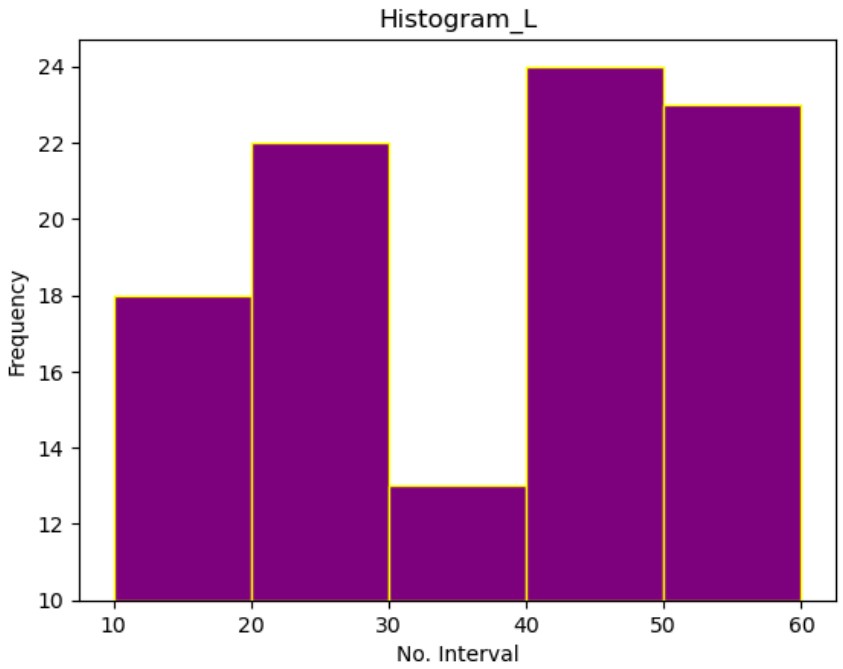
```
plt.hist(x,color="purple",bins=l,edgecolor='yellow',align='m
id',histtype="barstacked")
```



if bottom nothing

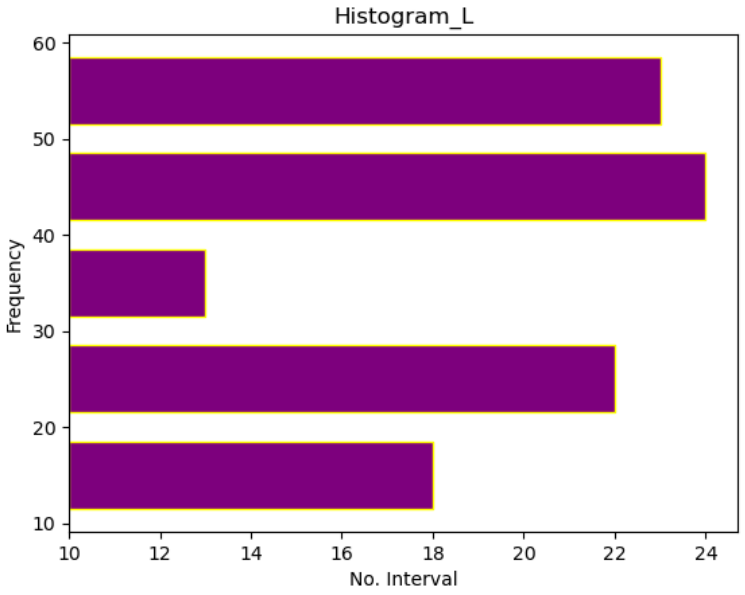
- **bar**

```
plt.hist(x,color="purple",bins=l,edgecolor='yellow',bottom=1
0,align='mid',histtype="bar")
```



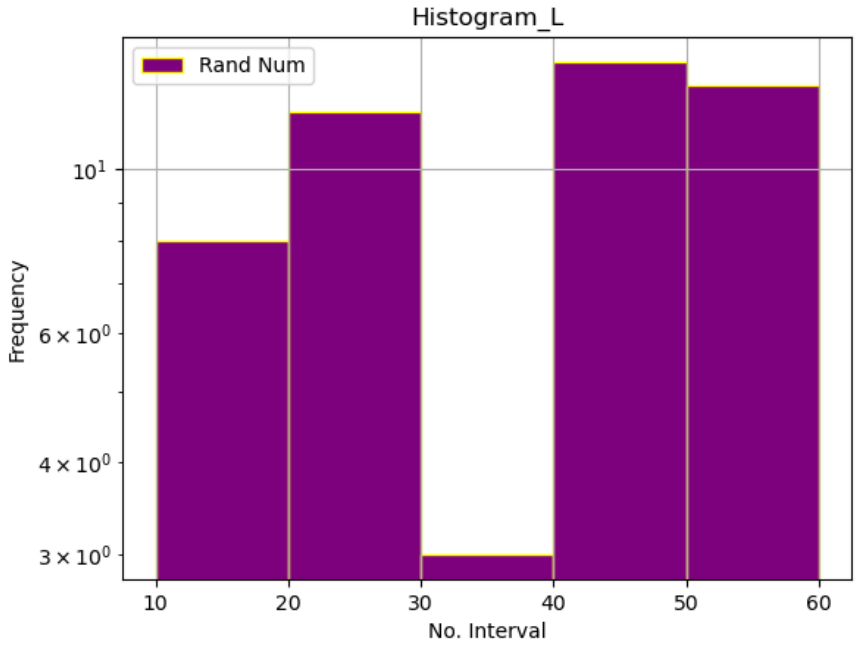
6) Orientation and rwidth

```
l = [10,20,30,40,50,60]
plt.hist(x,color="purple",bins=l,edgecolor='yellow',bottom=1
0,align='mid',orientation="horizontal",rwidth=0.7)
plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")
```



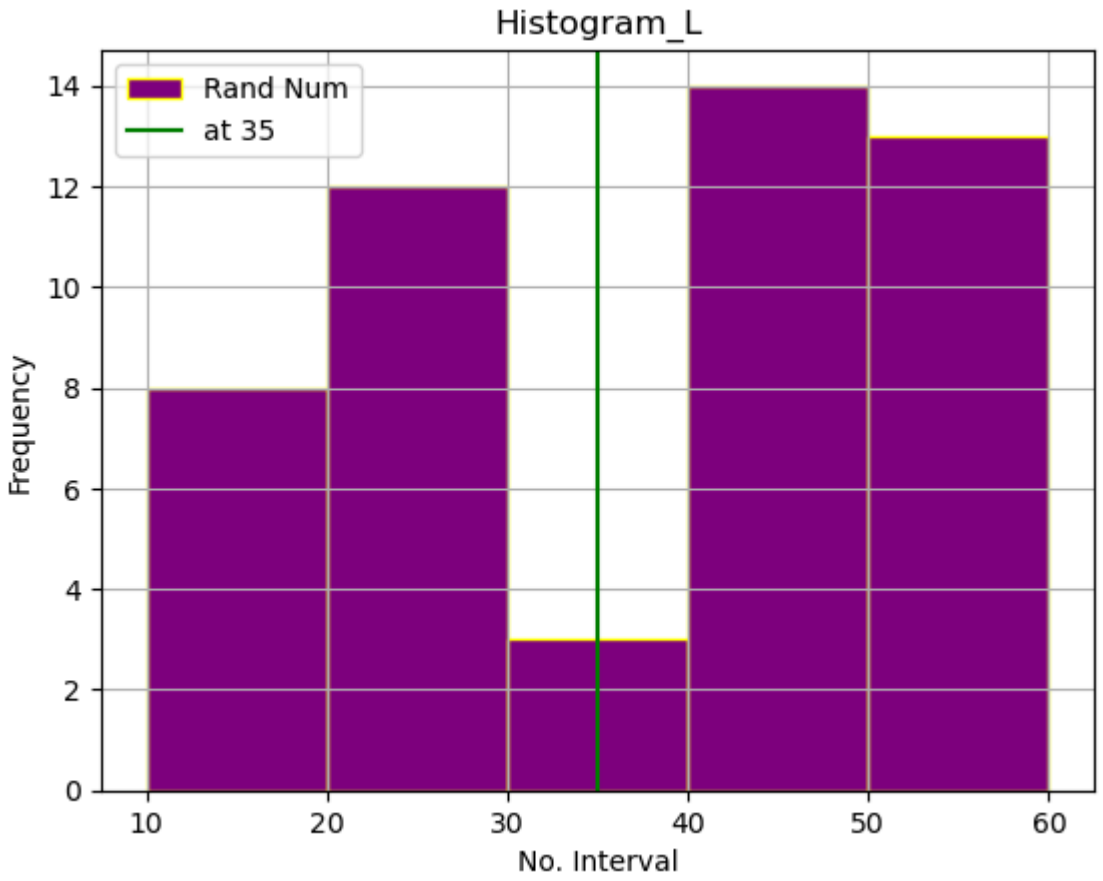
7) Log, Label (plt.legend() – mandatory for label), grid

```
l = [10,20,30,40,50,60]
plt.hist(x,color="purple",bins=l,edgecolor='yellow',align='m
id',log=True,label="Rand Num")
plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")

plt.legend()
plt.grid()
```



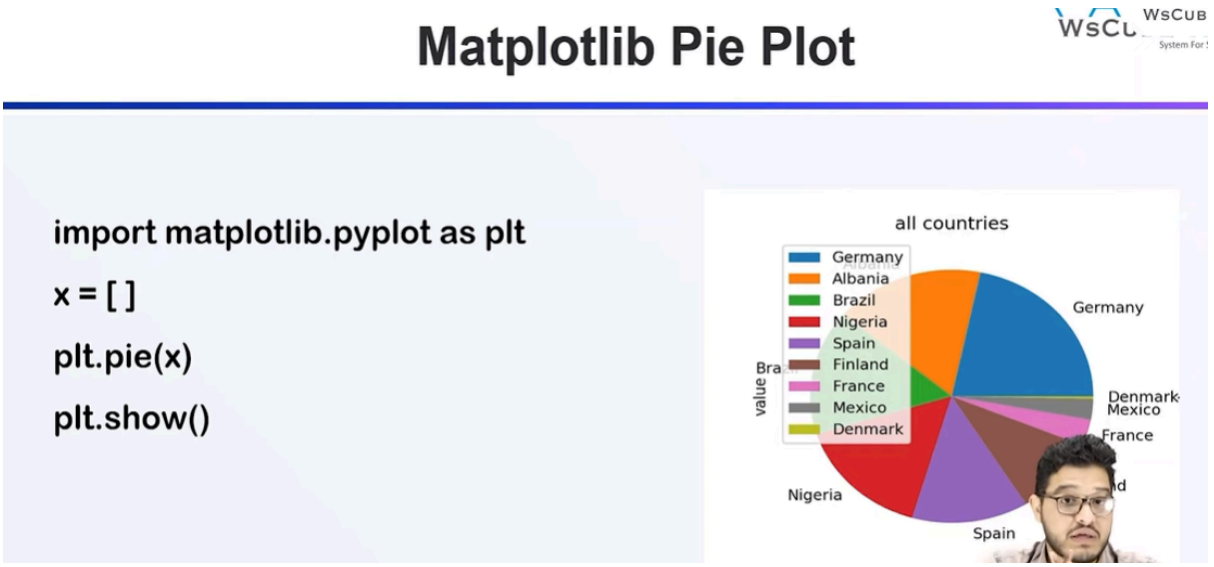
8) Axvline – a line created by us for analyse

```
l = [10,20,30,40,50,60]
plt.hist(x,color="purple",bins=l,edgecolor='yellow',align='m
id',label="Rand Num")
plt.title("Histogram_L")
plt.xlabel("No. Interval")
plt.ylabel("Frequency")

plt.grid()
plt.axvline(35,color="g",label="at 35")
plt.legend()
```


Histogram_L
Rand Num
at 35
Frequency
No. Interval
0
2
4
6
8
10
12
14
10
20
30
40
50
60

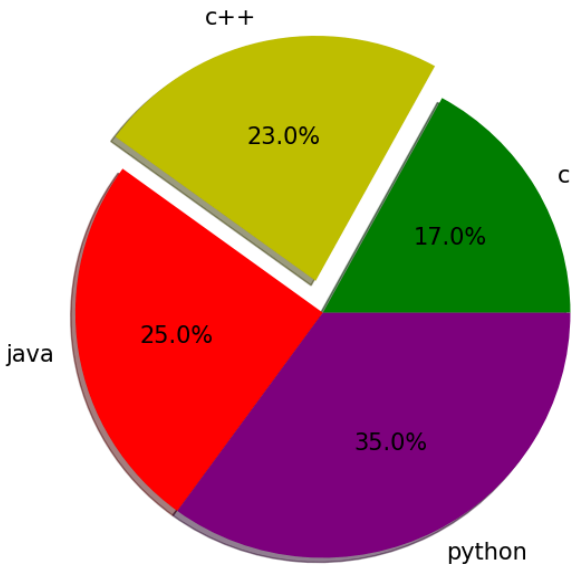
Pie graph plot

Pie graph plot



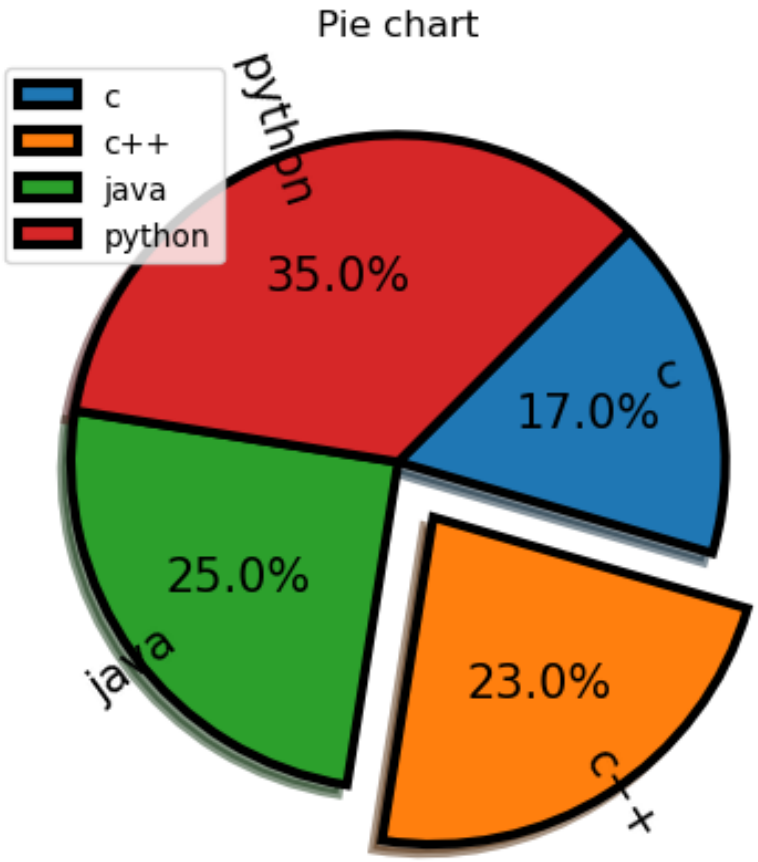
1) Normal,colour,ex,percentage,shadow,length,textsize

```
x = [17,23,25,35]
y = ["c","c++","java","python"]
c = ["g","y","r","purple"]
ex = [0.0,0.2,0.0,0.0]


plt.pie(x,labels=y,colors=c,explode=ex,autopct="%0.1f%%",sha
dow=True,radius=1.5,textprops={"fontsize":15})

plt.show()
```



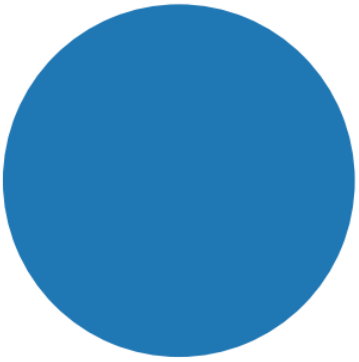
2) Labeldistance (distance b/w labels and pie),
Startangle - pie at which angle,
Counterclock - rotate pie clock wise (keep it - false)
Wedgeprops - properties for edge…
Title and legend (loc for - change legend place)

```
x = [17,23,25,35]
y = ["c","c++","java","python"]
ex = [0.0,0.2,0.0,0.0]

plt.pie(x,labels=y,explode=ex,autopct="%0.1f%%",shadow=True,
textprops={"fontsize":15},labeldistance=0.8,startangle=45,
counterclock=False,wedgeprops={"linewidth":3,"edgecolor":"bl
ack"},center=(0,0),rotatelabels="true")

plt.title("Pie chart",fontsize=15)
plt.legend(loc=2)
plt.show()
```



3) Dot pie chart

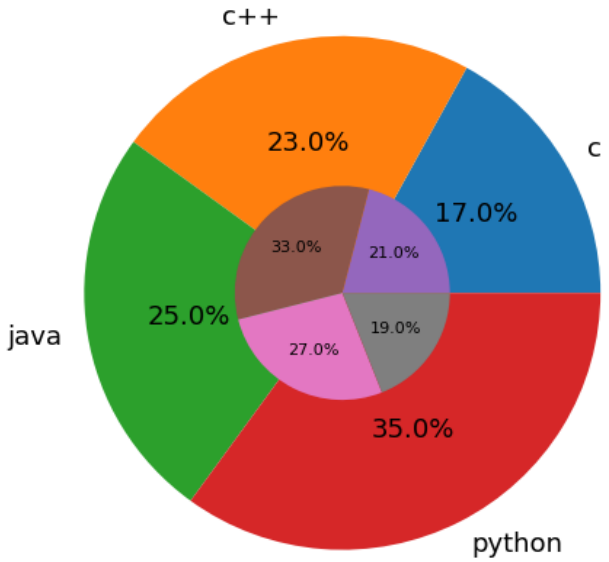
```
plt.pie([1])
plt.show()
```



4) Making two pie chart in one pie chart

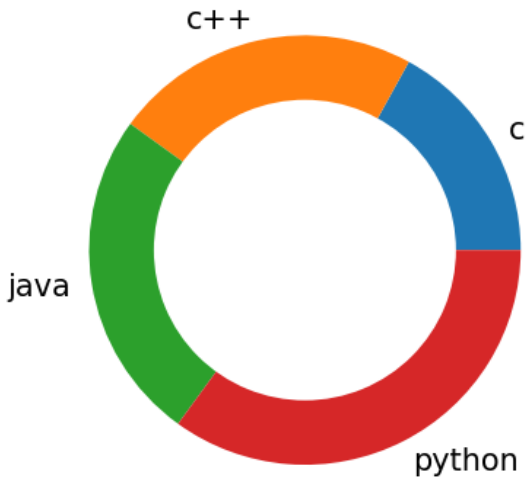
```
x = [17,23,25,35]
z = [21,33,27,19]
y = ["c","c++","java","python"]
```

```
plt.pie(x,labels=y,autopct="%0.1f%%",textprops={"fontsize":13},radius=1.2)
plt.pie(z,autopct="%0.1f%%",textprops={"fontsize":8},radius=0.5)
plt.show()
```



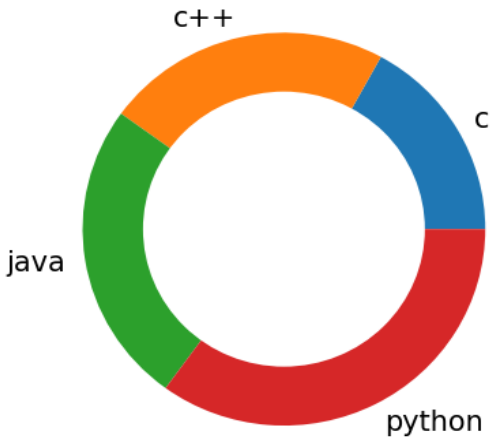
5) Donut pie chart

```
x = [17,23,25,35]
y = ["c","c++","java","python"]

plt.pie(x,labels=y,textprops={"fontsize":15})
plt.pie([1],radius=0.7,colors="w")
plt.show()
```



- And also another method —

```
a = [17,23,25,35]
b = ["c","c++","java","python"]
plt.pie(a,labels=b,textprops={"fontsize":15})
cr = plt.Circle(xy=(0,0),radius=0.7,facecolor="w")
plt.gca().add_artist(cr)

plt.show()
```

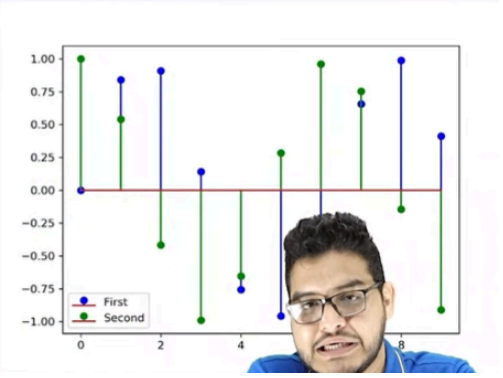


stem plot

Stem plot

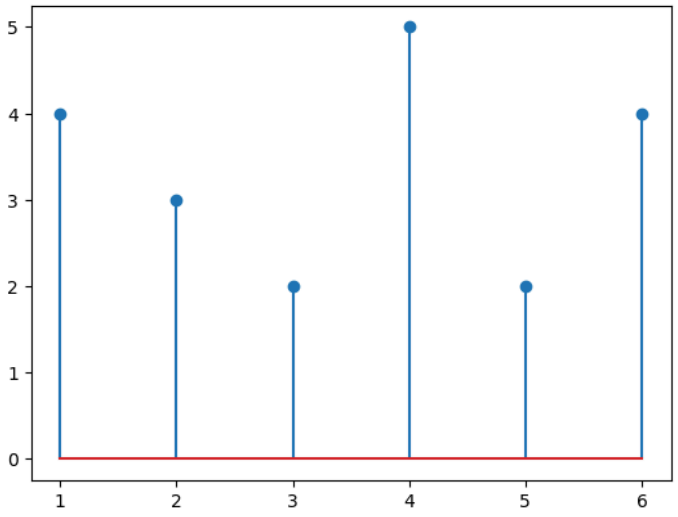
Matplotlib Stem Plot

```
import matplotlib.pyplot as plt
x = [ ]
y = [ ]
plt.stem(x,y)
plt.show()
```



1) Normal

```
x = [3,1,4,5,2,6]
y = [2,4,5,2,3,4]

plt.stem(x,y)
plt.show()
```

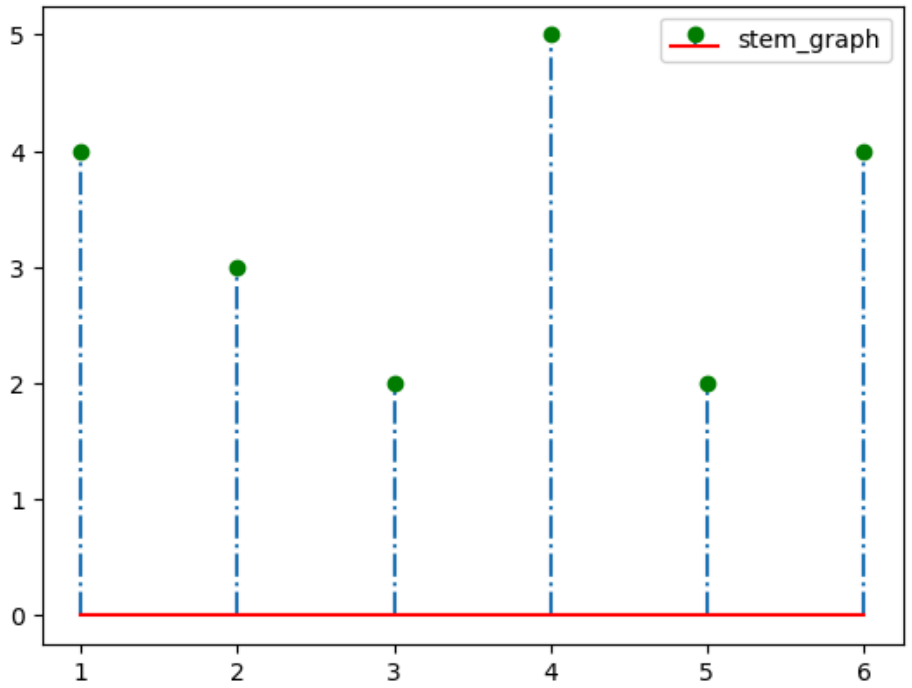


2) Modified

Linefmt :	character	line style
	```'-'```	Solid line
	```'--'```	Dashed line
	```'-.'```	Dash-dot line
	```':'```	Dotted line

Linefmt - for line style
Markerfmt - for point style ('+'or go,ro,vo (g,r - color))
Basefmt - for colour for below line
Label and legend

```
x = [3,1,4,5,2,6]
y = [2,4,5,2,3,4]

plt.stem(x,y,linefmt="-.",markerfmt="go",label="stem_graph",
basefmt="r")

plt.legend()
plt.show()
```

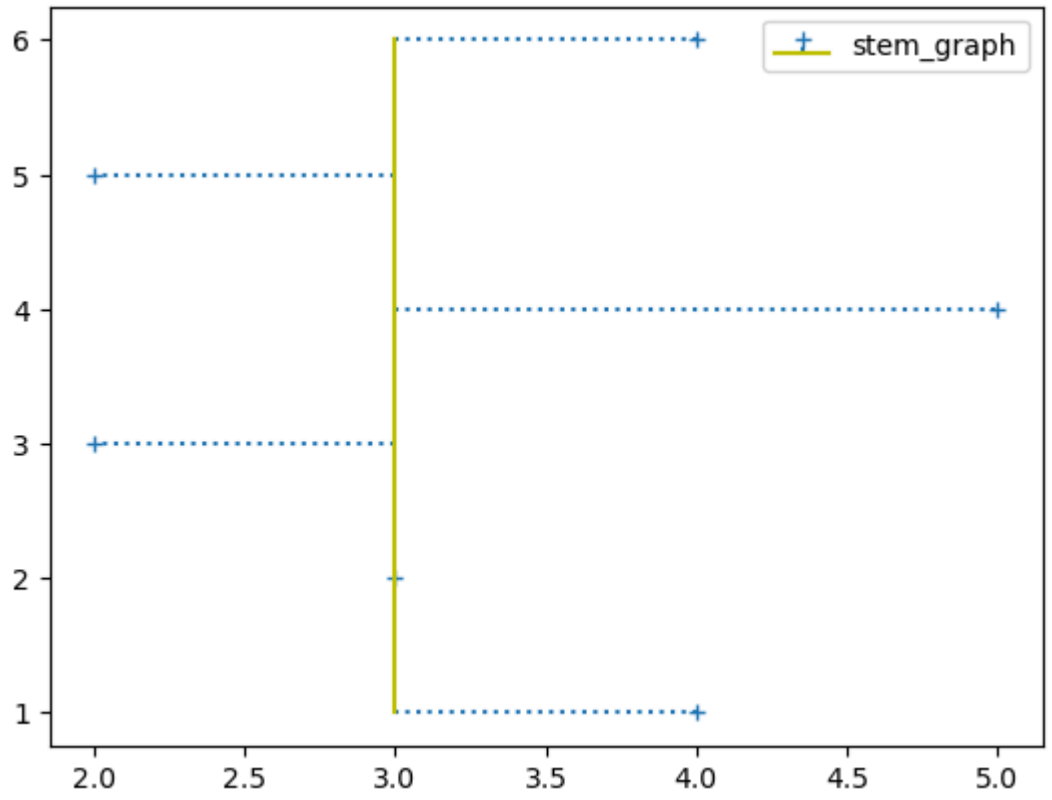


3) Modified 2 -

Bottom - it define base line
Orientation - vertical or horizontal
Use_line_collection - change colour of base line
But not available in latest matplotlib versions

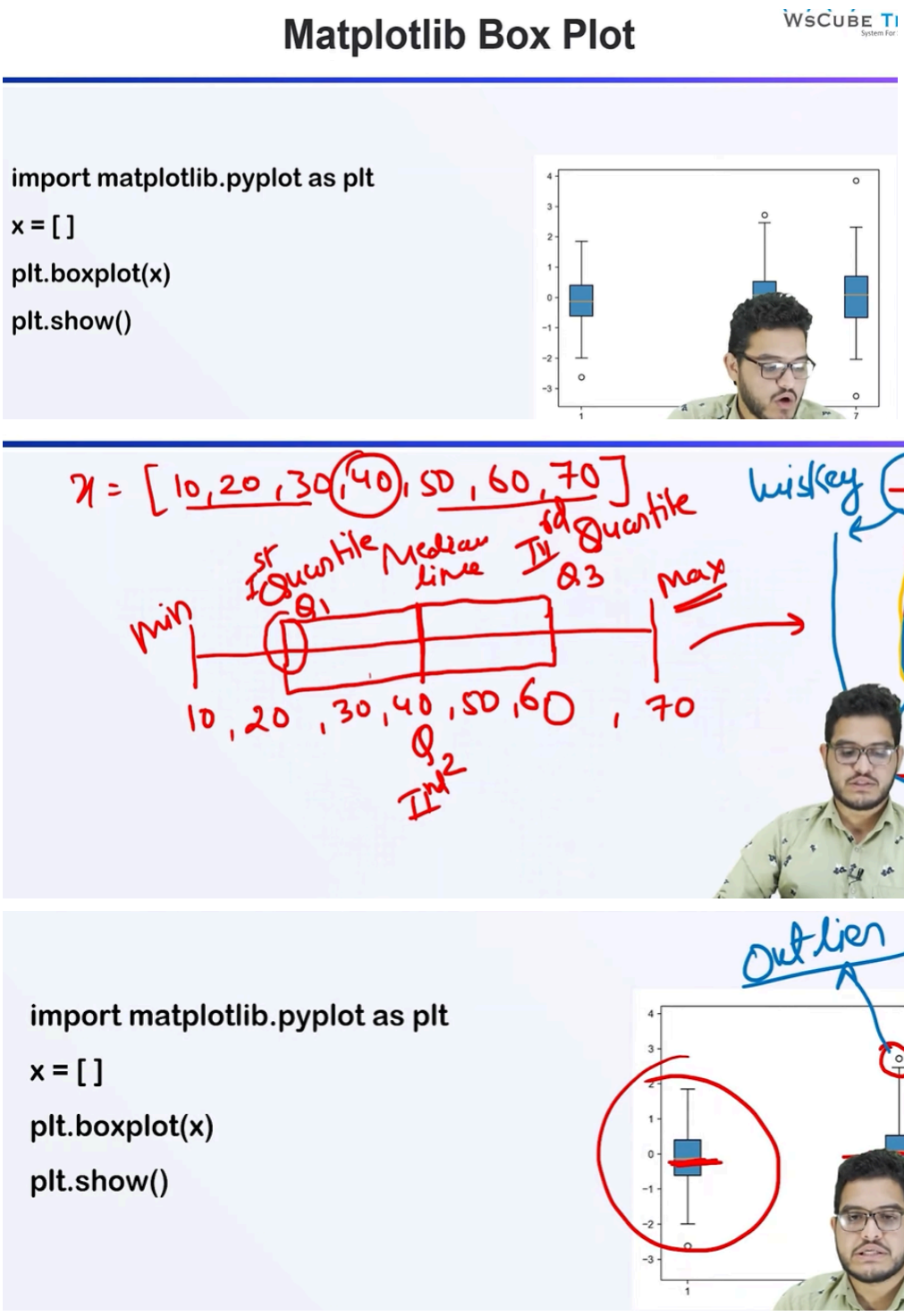
```
x = [3,1,4,5,2,6]
y = [2,4,5,2,3,4]

plt.stem(x,y,linefmt=":",markerfmt="+",label="stem_graph",ba
sefmt="y",bottom=3,orientation="horizontal")
```

```
plt.legend()
plt.show()
```



Box vs whisker

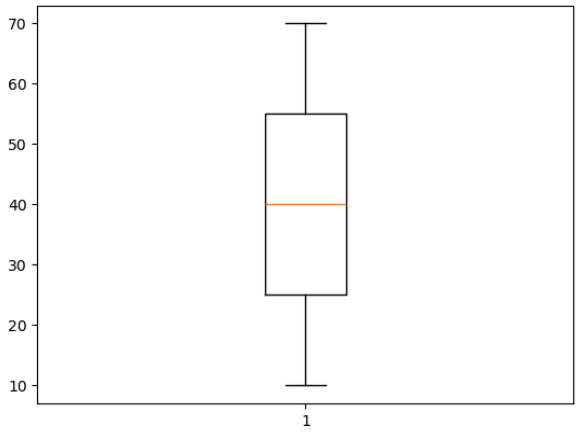
Box plot vs Whisker Plot



1) Normal

```
x = [10,20,30,40,50,60,70]
plt.boxplot(x)
```

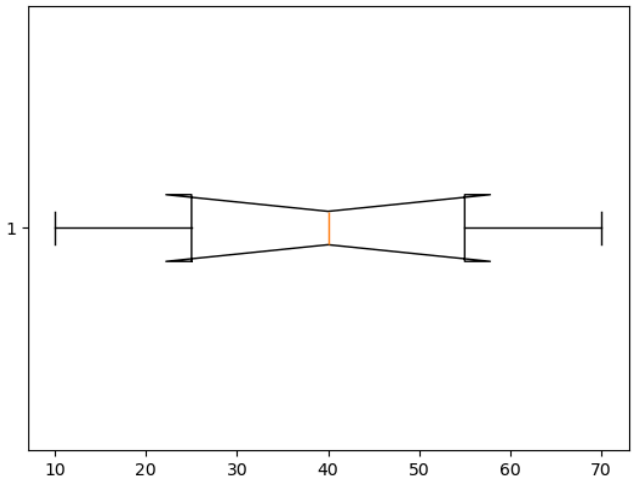
```
plt.show()
```



2) Modified -

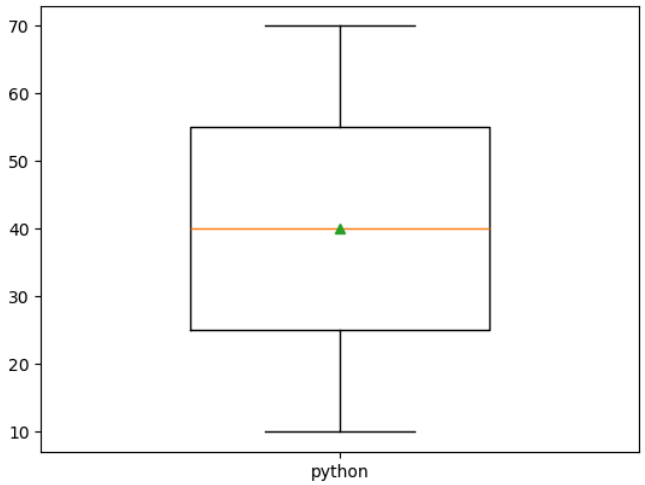
a) Notch and vertical

```
x = [10,20,30,40,50,60,70]

plt.boxplot(x,notch=True,vert=False)
plt.show()
```



b) Widths, labels, showmeans (point at center)

```
x = [10,20,30,40,50,60,70]

plt.boxplot(x,widths=0.5,labels=["python"],showmeans=T
rue)

plt.show()
```



c) Colour

Patch_artist - fill colour in box

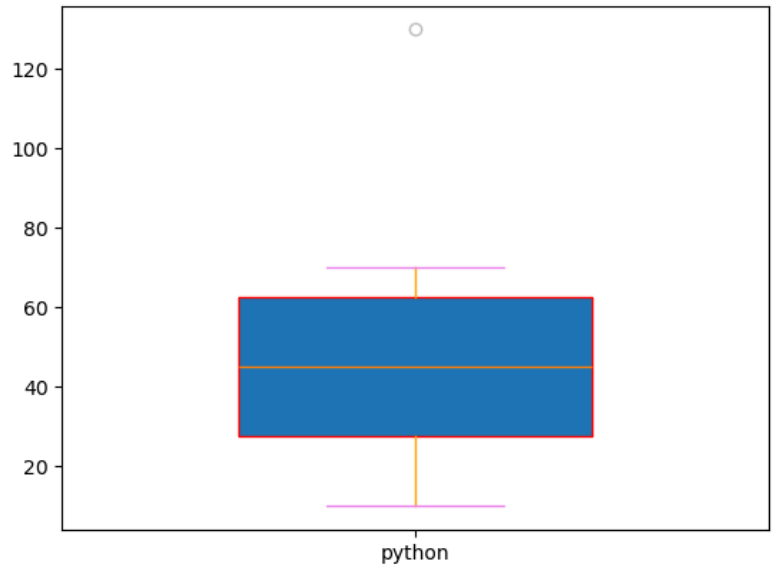
Boxprops - fill colr boundary of box

Capprops - fill colr or cap line up and down(violet)

Whiskerprops - straight line at cap(orange)

Flierprops - fill color of ouliner (silver)

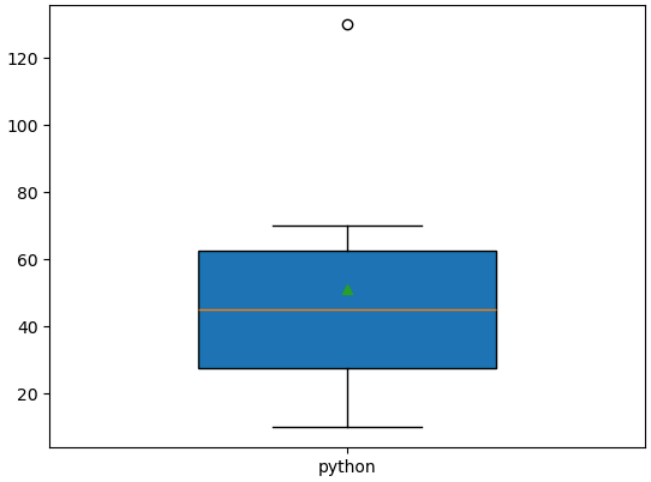
```
x = [10,20,30,40,50,60,70,130]

plt.boxplot(x,widths=0.5,labels=["python"],patch_artist=True,boxprops=dict(color="r"),capprops=dict(color="violet"),whiskerprops=dict(color="orange"),
flierprops=dict(markeredgecolor="silver"))

plt.show()
```



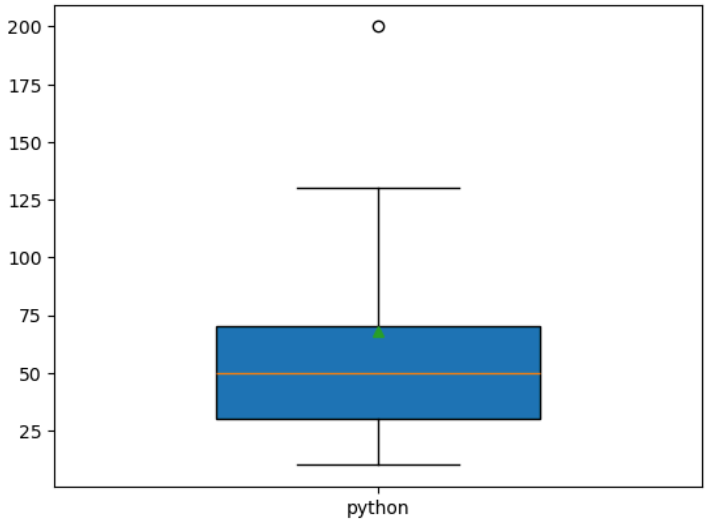
3) Outliner - (for outliner we have to keep one outliers data)

(so here is 130 outliers and if we take 200 also then 200 will outliers … outliers will be only one)

```
x = [10,20,30,40,50,60,70,130]
plt.boxplot(x,widths=0.5,labels=["python"],showmeans=True,patch_artist=True)
plt.show()
```

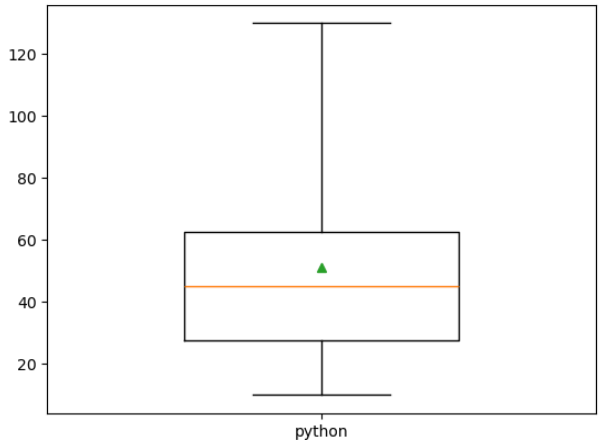



```
x = [10,20,30,40,50,60,70,130,200]
plt.boxplot(x,widths=0.5,labels=["python"],showmeans=True,patch_artist=True)
plt.show()
```



- **Modification in Outliner**
  - **a) Whis - if want to join outliers with whisker**

```
x = [10,20,30,40,50,60,70,130]

plt.boxplot(x,widths=0.5,labels=["python"],showmeans=T
rue,whis=2)
plt.show()
```

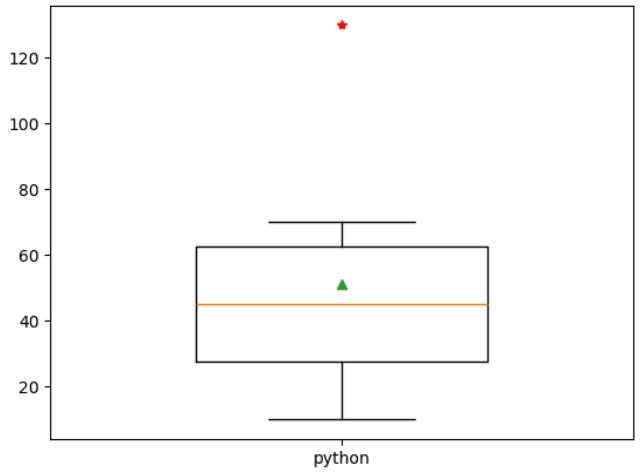


b) Sym - change outliers colour or shape

Like - g+ = (green and + shape), “ “ - means nothing (hide)

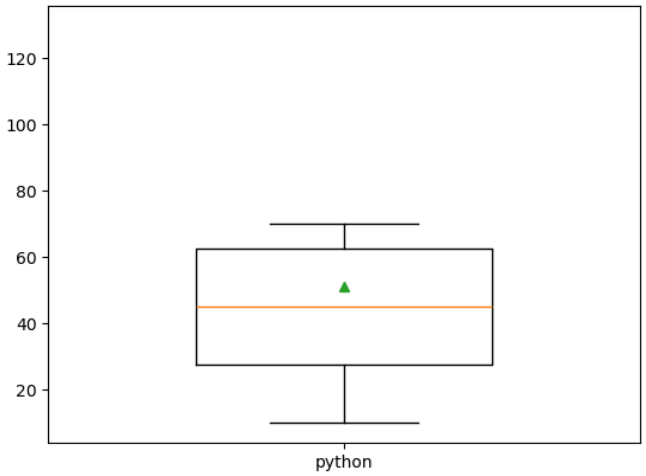
Eg 1-

```
x = [10,20,30,40,50,60,70,130]
plt.boxplot(x,widths=0.5,labels=["python"],showmeans=T
rue,sym="r*")
plt.show()
```



Eg 2- hide

```
x = [10,20,30,40,50,60,70,130]
plt.boxplot(x,widths=0.5,labels=["python"],showmeans=T
rue,sym=" ")
plt.show()
```



4) Multiplier Boxplot

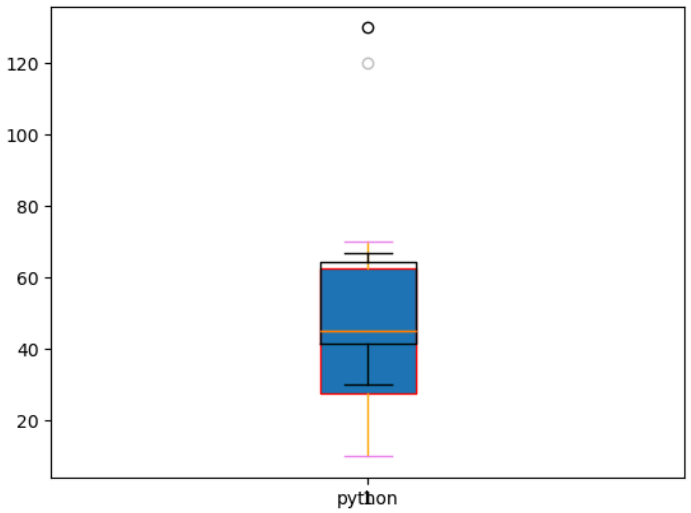
If i will try simple -

```
x = [10,20,30,40,50,60,70,120]
y = [40,62,43,30,45,67,130]
```

```
plt.boxplot(x,labels=["python"],patch_artist=True,boxprops=d
ict(color="r"),capprops=dict(color="violet"),whiskerprops=di
ct(color="orange"),flierprops=dict(markeredgecolor="silver")
)
```

```
plt.boxplot(y,label="c++")
```

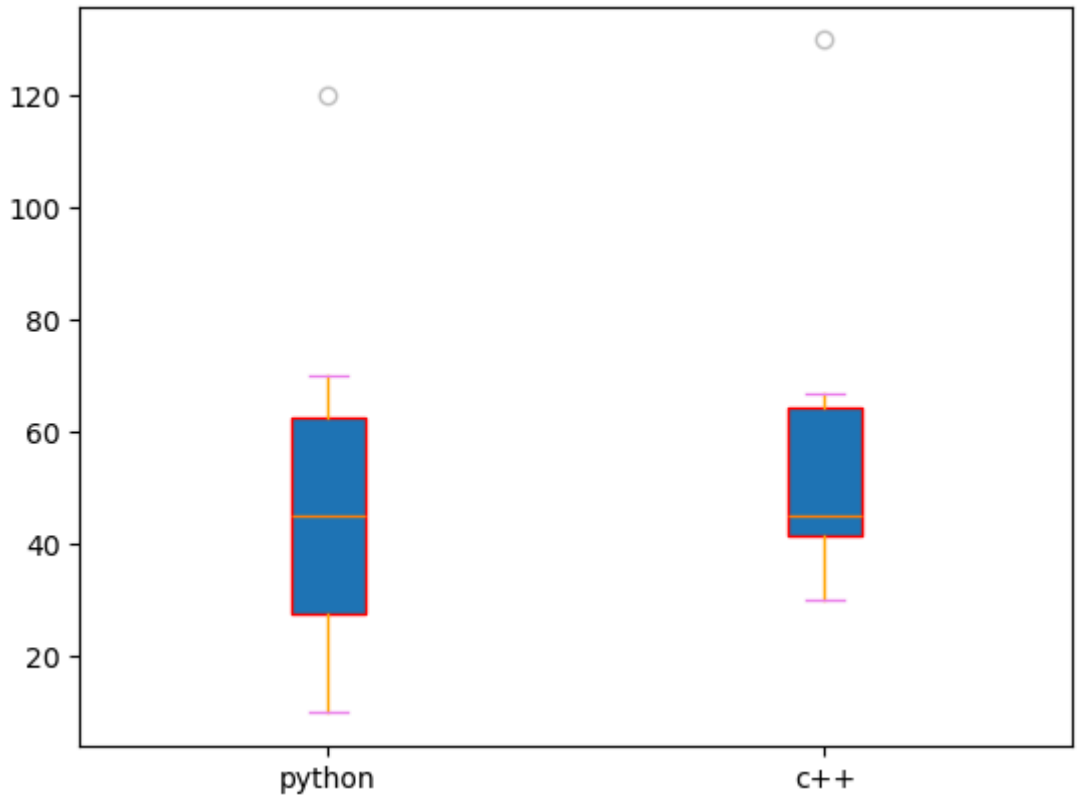
```
plt.show()
```



- Then both graph will show at one graph …

So , solution of it-

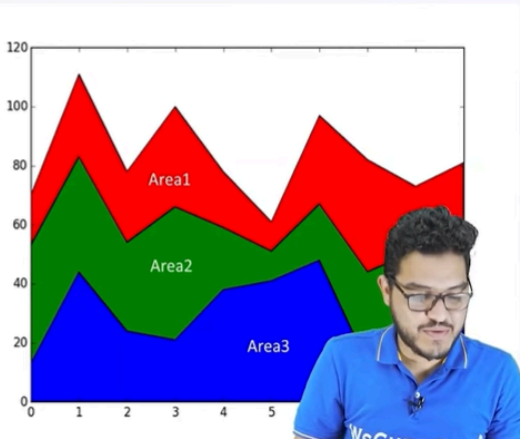
```
x = [10,20,30,40,50,60,70,120]
y = [40,62,43,30,45,67,130]

z = [x,y] #-- here it means lists.. For labels …

plt.boxplot(z,labels=["python","c++"],patch_artist=True,boxprops=dict(color="r"),capprops=dict(color="violet"),whiskerprops=dict(color="orange"),flierprops=dict(markeredgecolor="silver"))

plt.show()
```



Stack Plot

Stack Plot

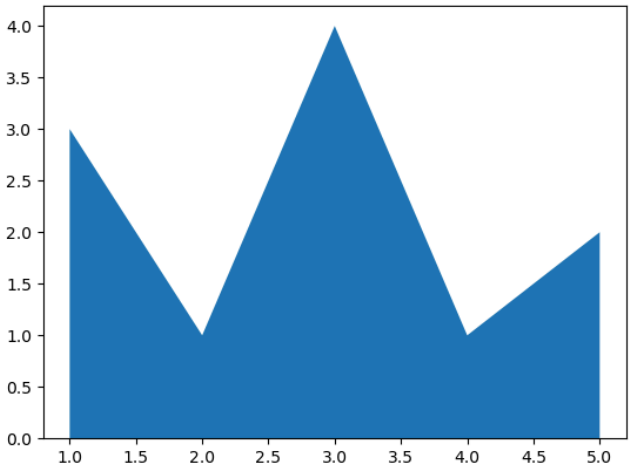
```
import matplotlib.pyplot as plt
x = [ ]
y = [ ]
plt.stackplot(x,y)
plt.show()
```



1) Normal

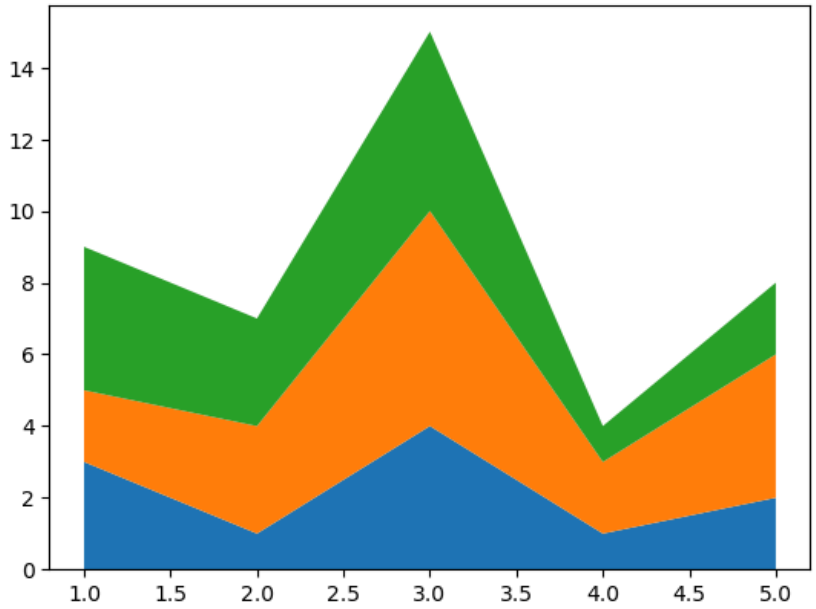
- **One area stack graph**

```
x = [1,2,3,4,5]
area = [3,1,4,1,2]

plt.stackplot(x,area)
plt.show()
```



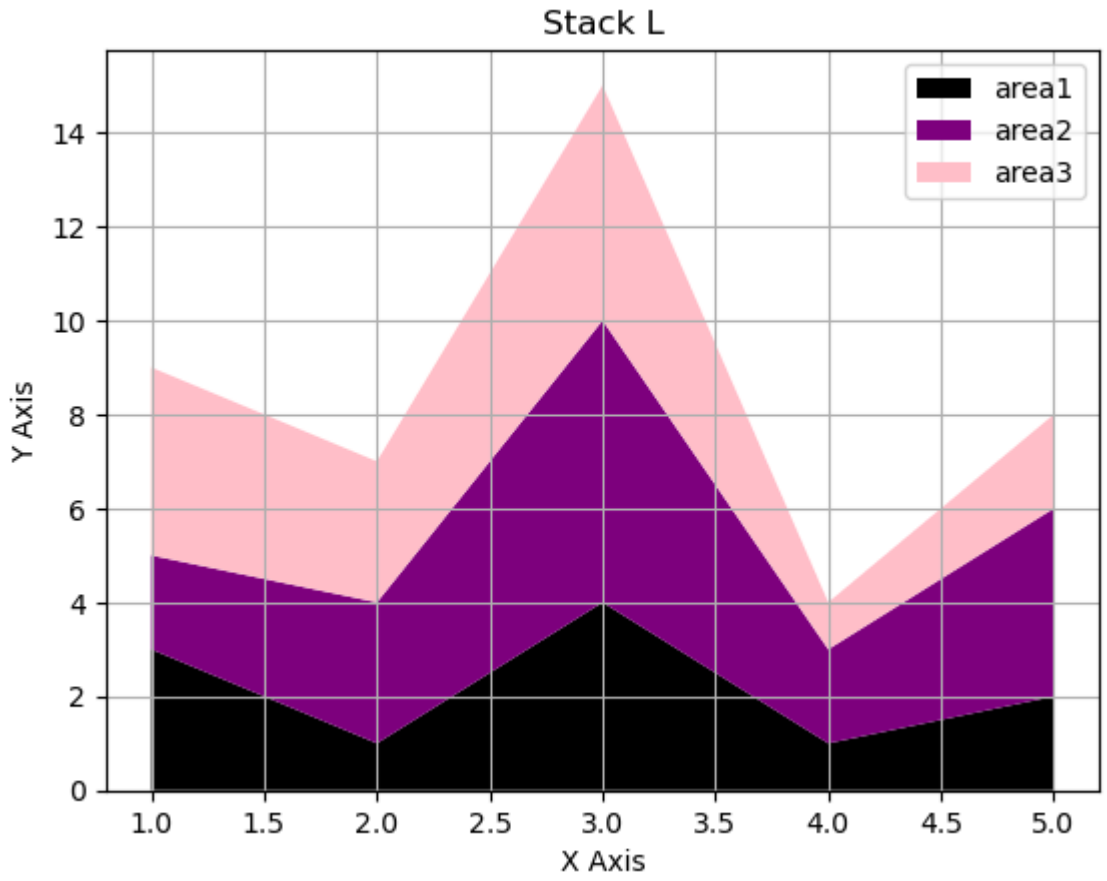
- **Multiple area stack graph**

```
x = [1,2,3,4,5]
area = [3,1,4,1,2]
area2 = [2,3,6,2,4]
area3 = [4,3,5,1,2]

plt.stackplot(x,area,area2,area3)
plt.show()
```



2) Modification

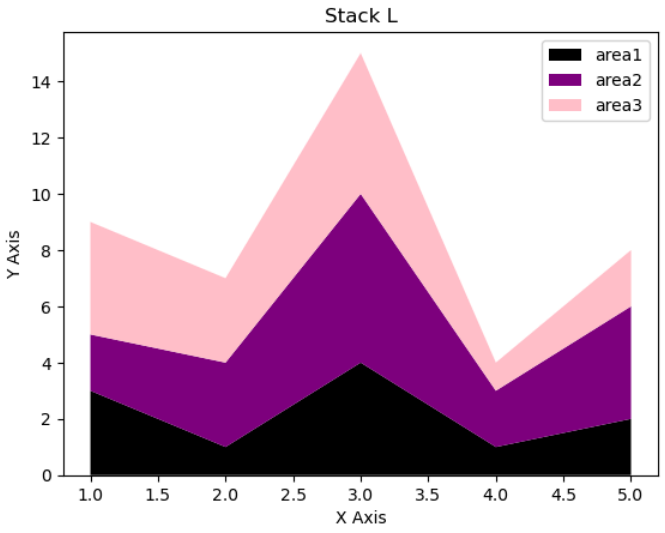
```
x = [1,2,3,4,5]
area1 = [3,1,4,1,2]
area2 = [2,3,6,2,4]
area3 = [4,3,5,1,2]

l = ["area1","area2","area3"]

plt.stackplot(x,area,area2,area3,labels=l,colors=["black","p
urple","pink"])

plt.title("Stack L")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.legend()
plt.grid()
plt.show()
```



3) Baseline

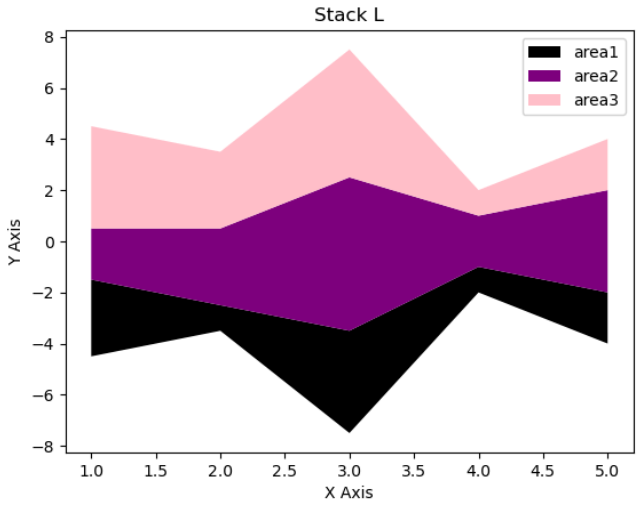
a) Zero or by default

```
plt.stackplot(x,area,area2,area3,labels=l,colors=["black","purple","pink"],baseline="zero")
```



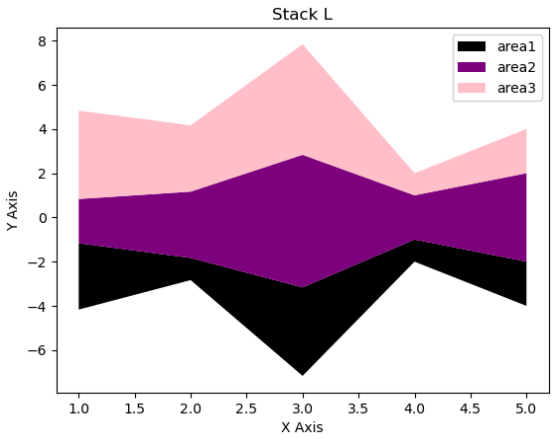
b) Sym

```
x = [1,2,3,4,5]
area1 = [3,1,4,1,2]
area2 = [2,3,6,2,4]
area3 = [4,3,5,1,2]
l = ["area1","area2","area3"]

plt.stackplot(x,area,area2,area3,labels=l,colors=["black","purple","pink"],baseline="sym")

plt.title("Stack L")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.legend()
plt.show()
```



c) wiggle

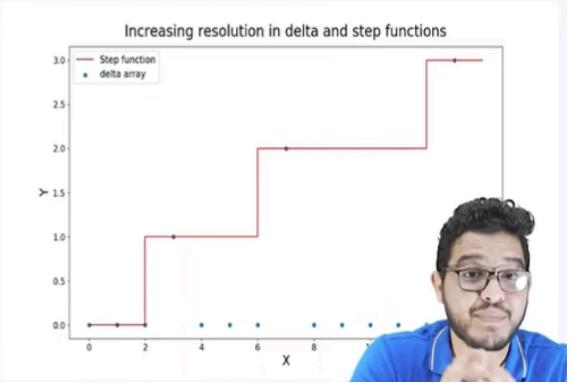
```
plt.stackplot(x,area,area2,area3,labels=l,colors=["black","purple","pink"],baseline="wiggle")
```



Step plot

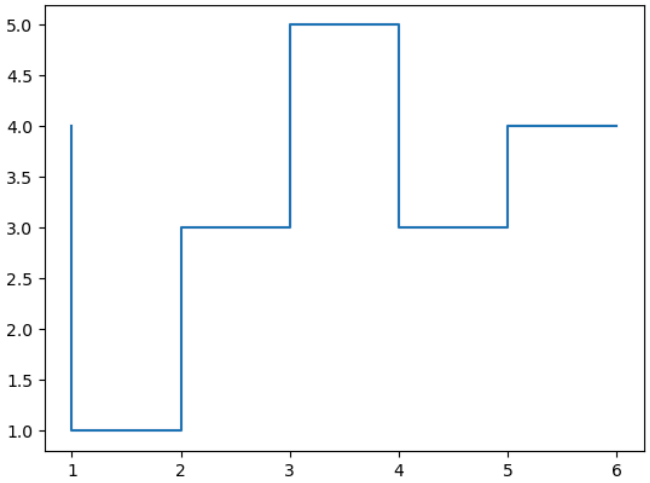
Step Plot

```
import matplotlib.pyplot as plt
x = [ ]
y = [ ]
plt.step(x,y)
plt.show()
```



1) Normal

```
x = [1,2,3,4,5,6]
y = [4,1,3,5,3,4]

plt.step(x,y)
plt.show()
```

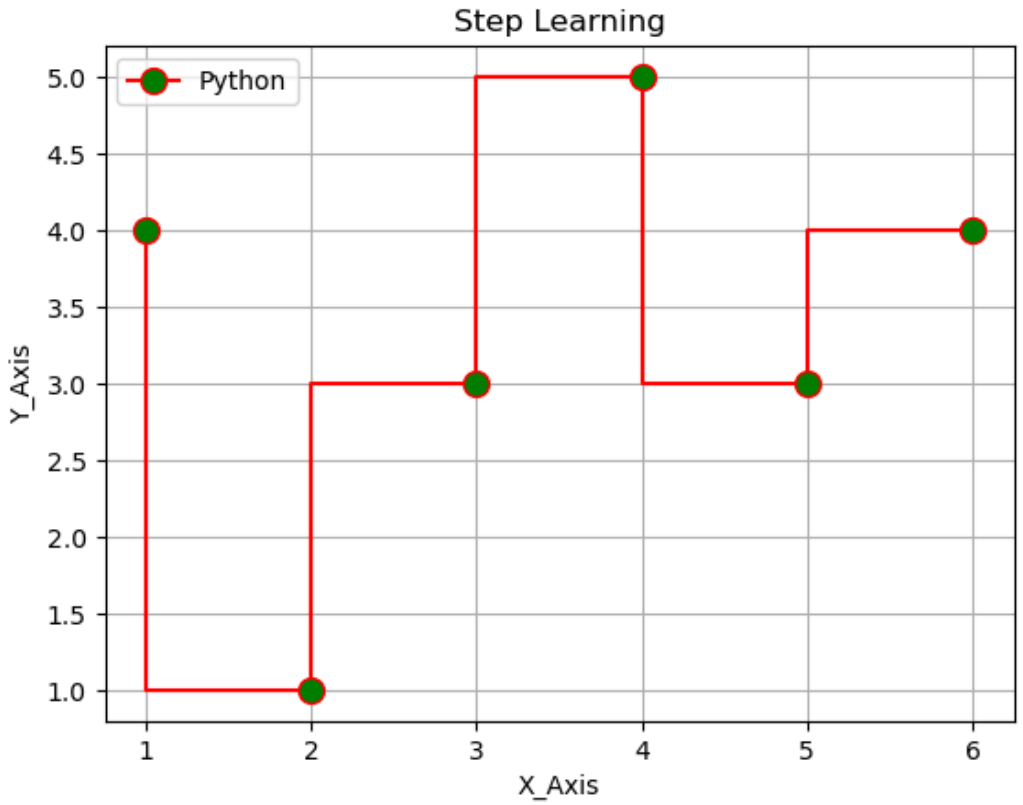


2) Modification

Color,marker, ms(marker or point area), mfc (point colr)

```
x = [1,2,3,4,5,6]
y = [4,1,3,5,3,4]

plt.step(x,y,color="r",marker="o",ms=10,mfc="green",label="P
ython")

plt.title("Step Learning")
plt.xlabel("X_Axis")
plt.ylabel("Y_Axis")
plt.legend(loc=2)
```

```
plt.grid()
plt.show()
```



3) Rest all features are same as line graph … notes

Subplot

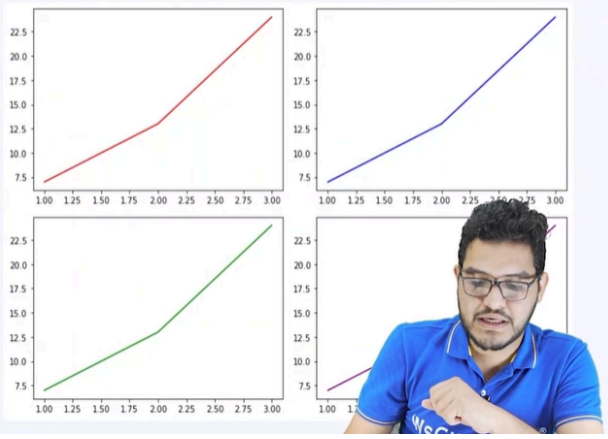
SubPlot

- Its means simple show many graph together

Matplotlib Subplot Plot

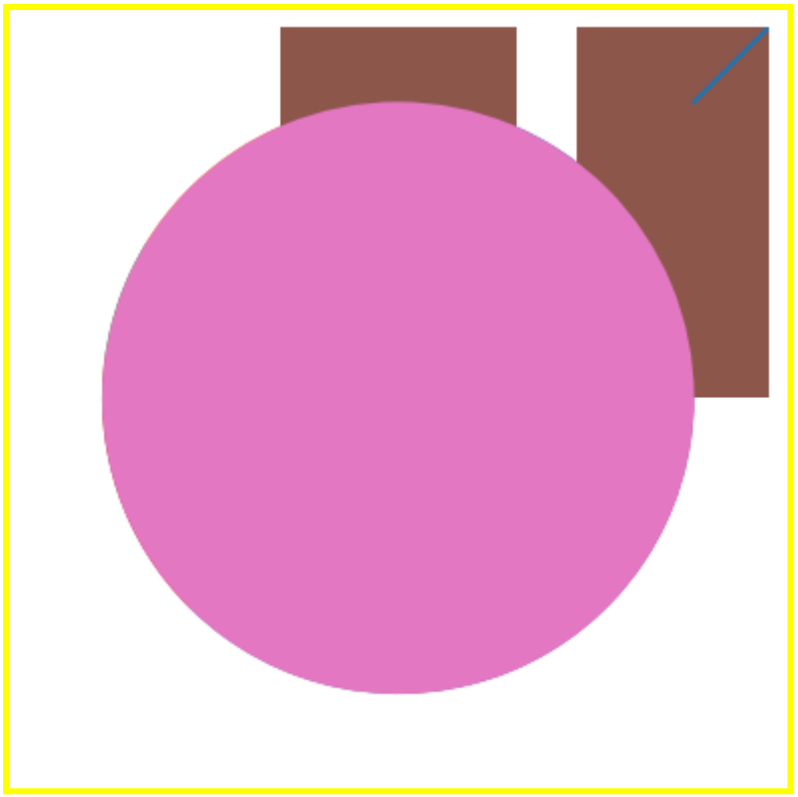
WsCube Tech
System For Satisfaction

```
import matplotlib.pyplot as plt
x = [ ]
y = [ ]
plt.subplot( nrows, ncols, index )
plt. plot(x,y)
plt.show()
```



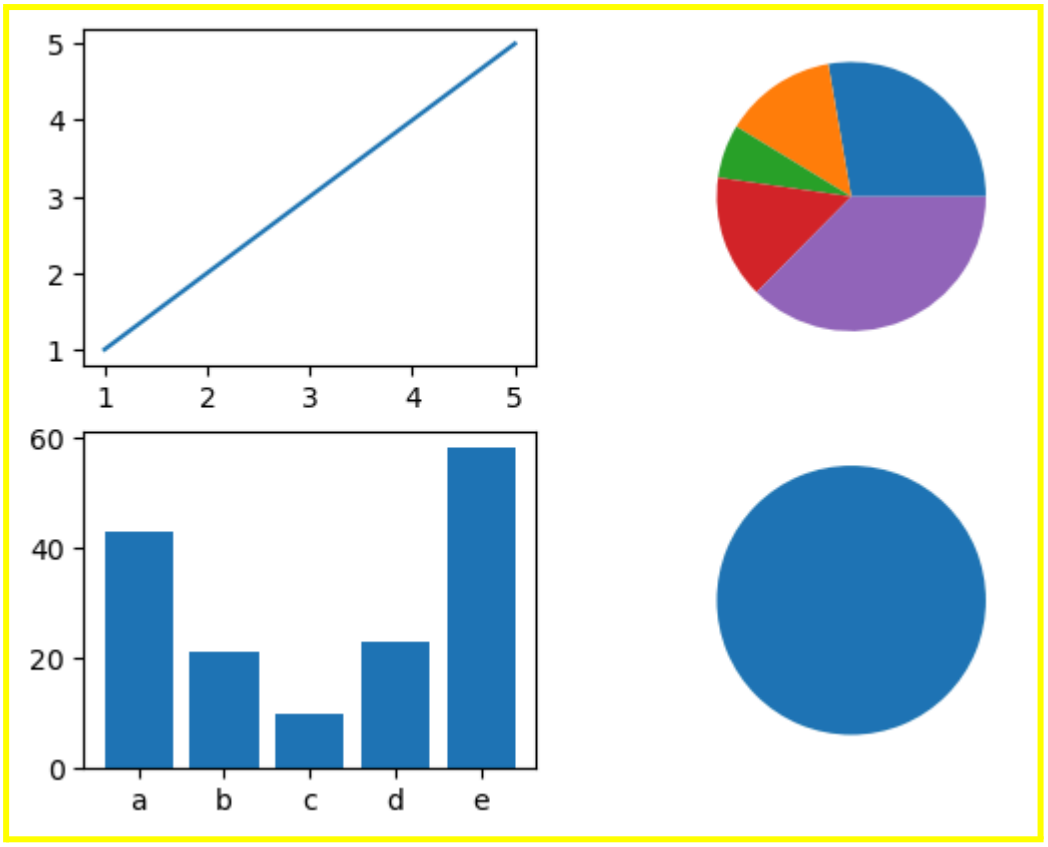
- Without Subplot

```
x = [1,2,3,4,5]
y = [1,2,3,4,5]

plt.plot(x,y)

pie_data = [43,21,10,23,58]
plt.pie(pie_data)

z = ["a","b","c","d","e"]
plt.bar(z,pie_data)

plt.pie([1])

plt.show()
```



- **With Subplot**

```
x = [1,2,3,4,5]
y = [1,2,3,4,5]

plt.subplot(2,2,1)
plt.plot(x,y)

pie_data = [43,21,10,23,58]
plt.subplot(2,2,2)
plt.pie(pie_data)

z = ["a","b","c","d","e"]
plt.subplot(2,2,3)
plt.bar(z,pie_data)

plt.subplot(2,2,4)
plt.pie([1])

plt.show()
```



- **why here subplot use only -- (2,2,1) or (2,2,2).. Etc**

What does `plt.subplot(2,2,1)` mean?

`plt.subplot(r, c, i)` has three parts:

(r, c, i)

Part	Meaning
r	number of rows
c	number of columns
i	position number (index)

So:
plt.subplot(2, 2, 1)

means:

> Create a figure divided into 2 rows × 2 columns, and activate subplot #1

> Why only `(2,2,1)` to `(2,2,4)`?

Because a 2 × 2 grid has only 4 slots:

> (2,2,1) (2,2,2)
> (2,2,3) (2,2,4)

If you try:

> plt.subplot(2,2,5)

❌ You'll get an error, because position 5 does not exist.

> How Matplotlib numbers subplots

Matplotlib numbers subplots left to right, top to bottom, like reading a book:

> 1 2
> 3 4

That's why your code works perfectly:

```
plt.subplot(2,2,1)  # top-left
plt.subplot(2,2,2)  # top-right
plt.subplot(2,2,3)  # bottom-left
plt.subplot(2,2,4)  # bottom-right
```

> Why not use something like `(2,2,0)` or `(2,2,10)`?

- Subplot indexing starts at 1, not 0
- Maximum index = `rows × columns`

So valid indices here are:

1, 2, 3, 4

Shortcut syntax (optional tip)

Matplotlib also allows this shorthand:

```
plt.subplot(221)  # same as plt.subplot(2,2,1)
```

Digits mean: `2 rows`, `2 columns`, `1st position`

Summary

✔ `subplot(2,2,i)` → grid of 2×2
✔ `i` must be between `1` and `4`
✔ Numbering is left → right, top → bottom
✔ That's why you only see `(2,2,1)` to `(2,2,4)`

If you want, I can also show how this works visually or explain `plt.subplots()` (the modern way). 😊

Savefig

Savefig

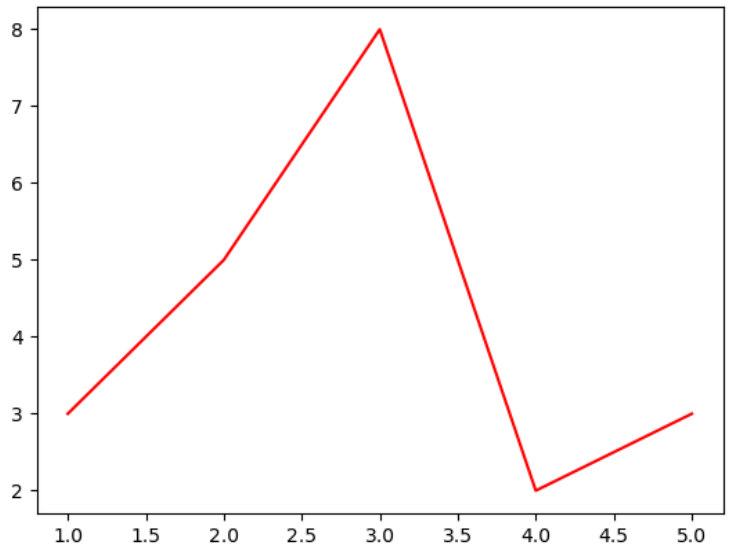
Matplotlib Savefig

```
import matplotlib.pyplot as plt
x = [ ]
y = [ ]
plt. plot(x,y)
plt.savefig( fname )
plt.show()
```

- It will save on that folder in which that file is written in that folder.

1) Normal

```
x = [1,2,3,4,5]
y = [3,5,8,2,3]

plt.plot(x,y,color="r")
plt.savefig("Line Graph")

plt.show()
```



- It will save in same file in which code written…

2) save file… in good quality and with some modification

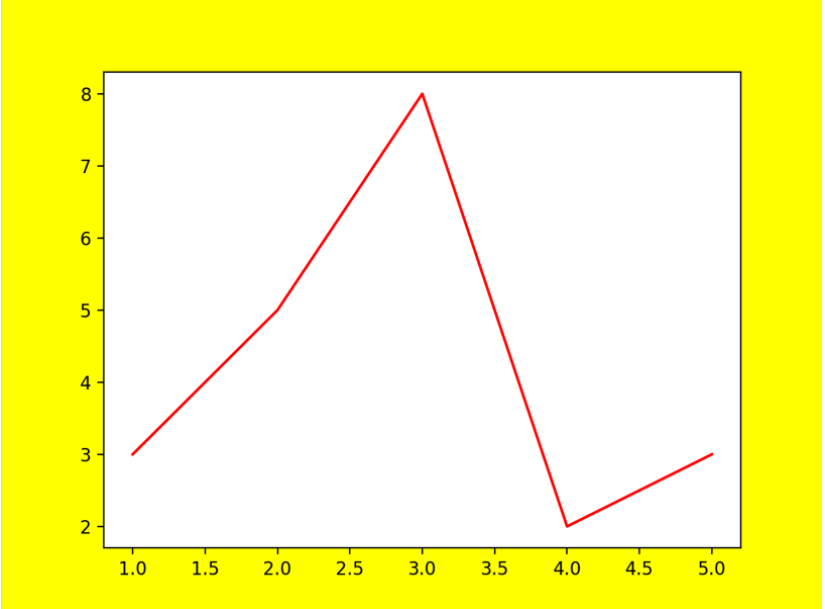
a) Dpi = Dot per inche

```
x = [1,2,3,4,5]
y = [3,5,8,2,3]

plt.plot(x,y,color="r")
plt.savefig("Line Graph",dpi=2000)

plt.show()
```

Output - Photo save in high quality photo, where 2000 dot in just 1 inch. (check matplotlib folder)

b) Facecolor

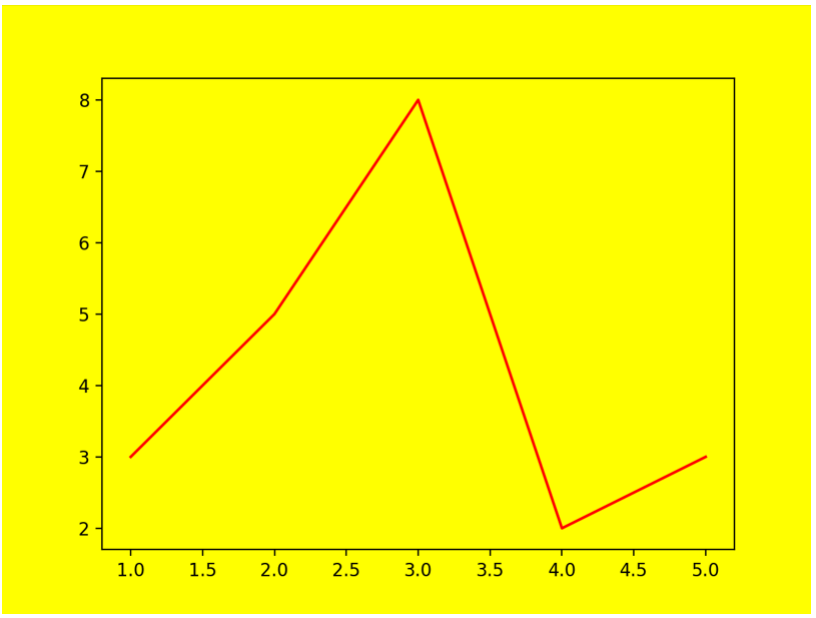
```
x = [1,2,3,4,5]
y = [3,5,8,2,3]

plt.plot(x,y,color="r")
plt.savefig("Line Graph1",dpi=200,facecolor="yellow")

plt.show()
```



c) Transparent

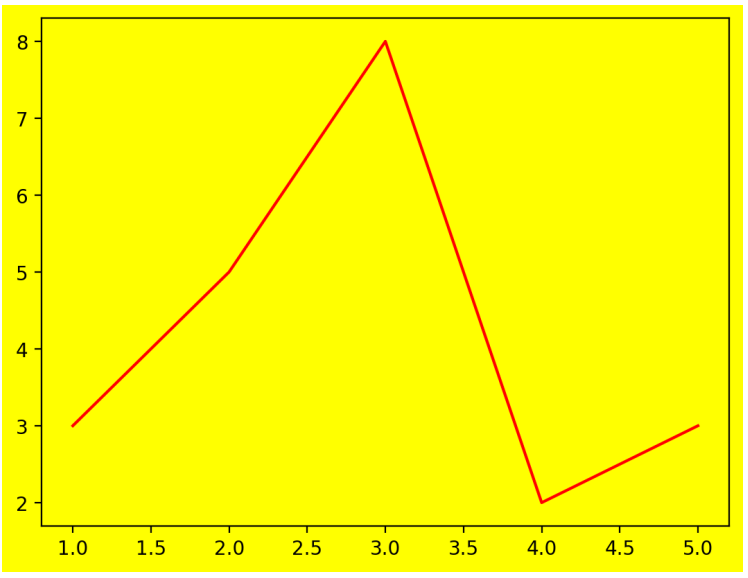
```
x = [1,2,3,4,5]
y = [3,5,8,2,3]
```

```
plt.plot(x,y,color="r")
plt.savefig("Line
Graph11",dpi=200,facecolor="yellow",transparent=True)

plt.show()
```



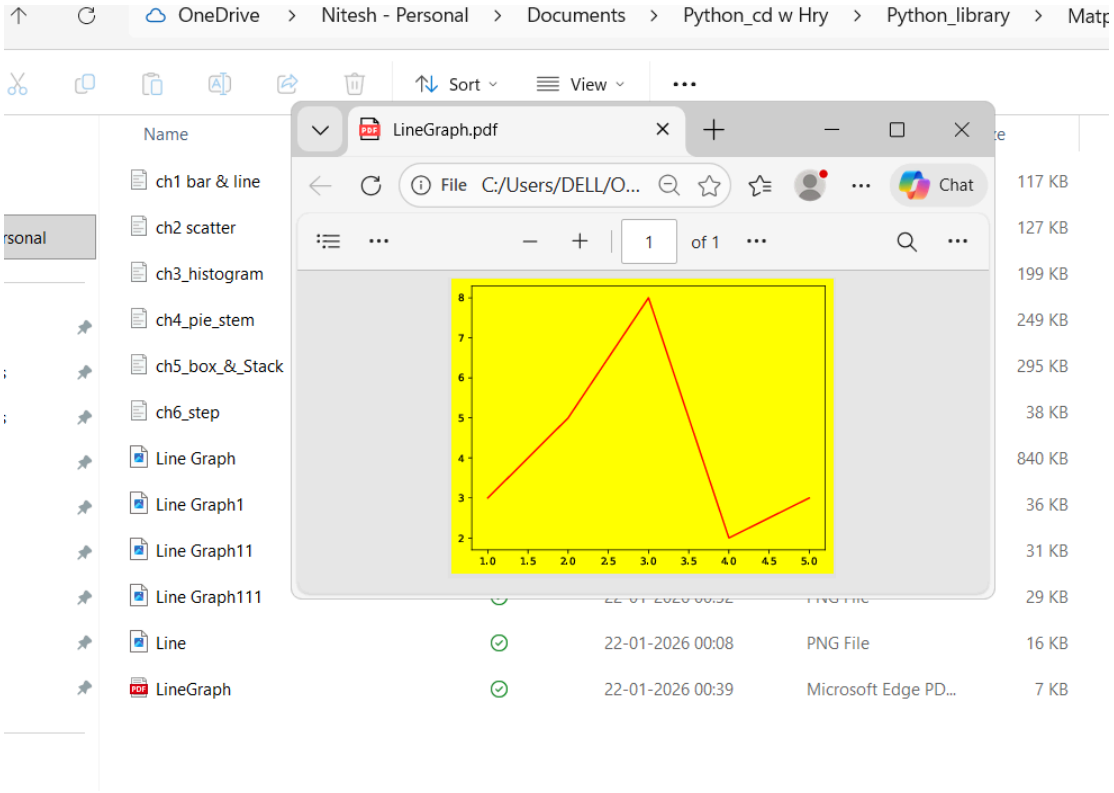
d) bbox_inches

```
x = [1,2,3,4,5]
y = [3,5,8,2,3]
plt.plot(x,y,color="r")
plt.savefig("Line
Graph111",dpi=200,facecolor="yellow",transparent=True,
bbox_inches="tight")
plt.show()
```



e) Save as pdf or png

```
x = [1,2,3,4,5]
y = [3,5,8,2,3]

plt.plot(x,y,color="r")
plt.savefig("LineGraph.pdf",dpi=200,facecolor="yellow"
,transparent=True,bbox_inches="tight")

plt.show()
```

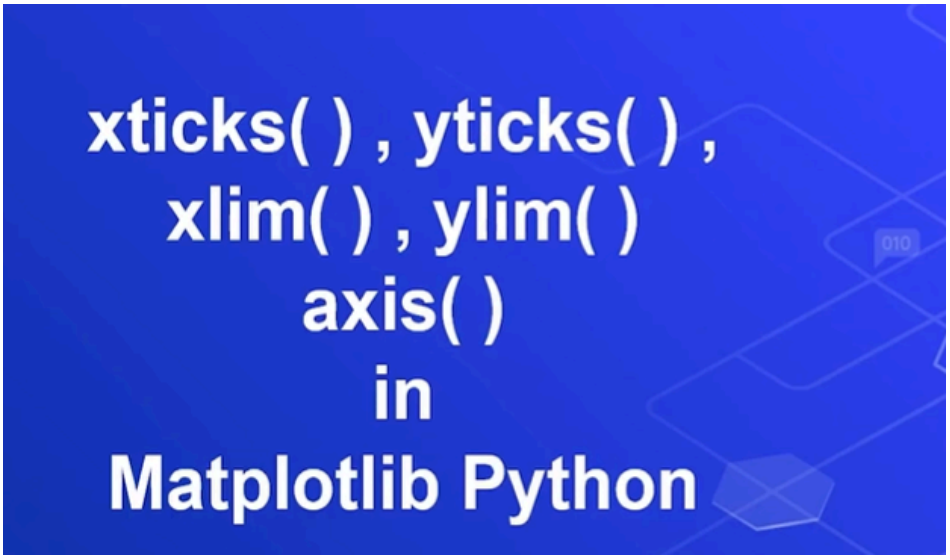


— same for png or any other… extension

```
plt.savefig("LineGraph.png",dpi=200,facecolor="yellow"
,transparent=True,bbox_inches="tight")
```

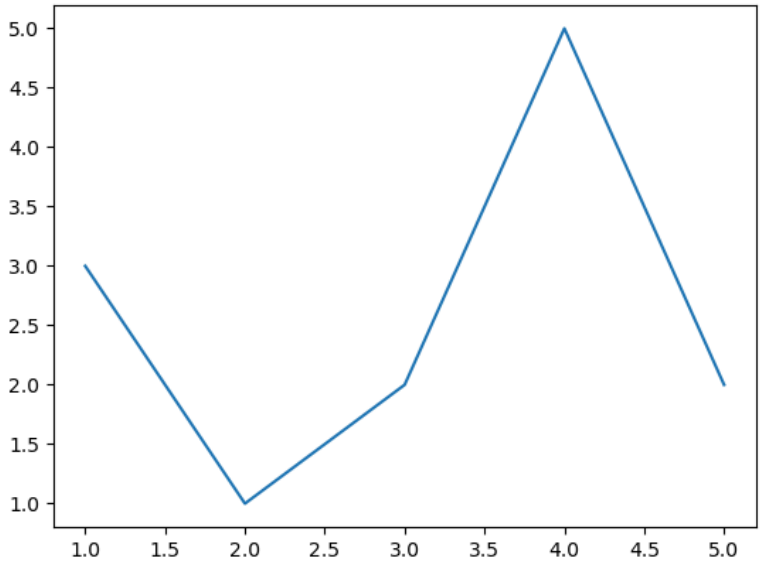
Axis Modify

Axis Modification



1) Normal

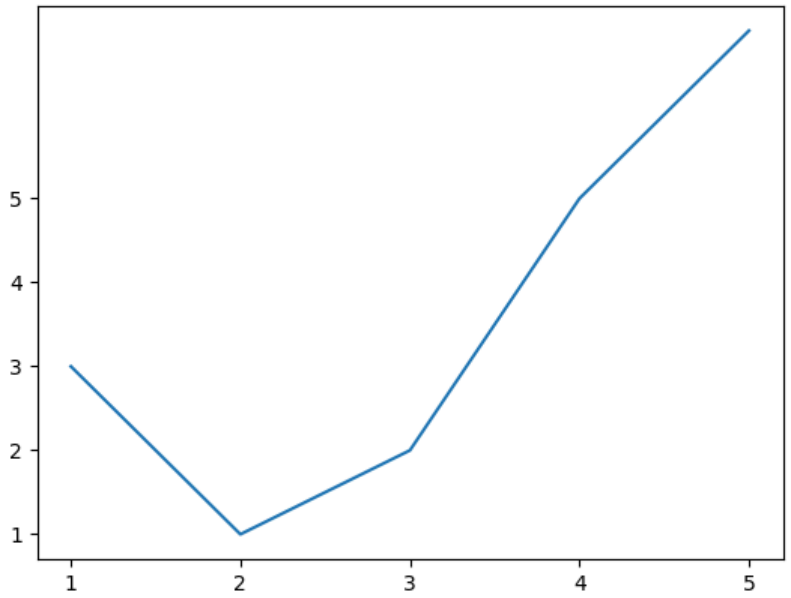
```
x = [1,2,3,4,5]
y = [3,1,2,5,2]

plt.plot(x,y)
plt.show()
```



2) After axis modification - xticks and yticks

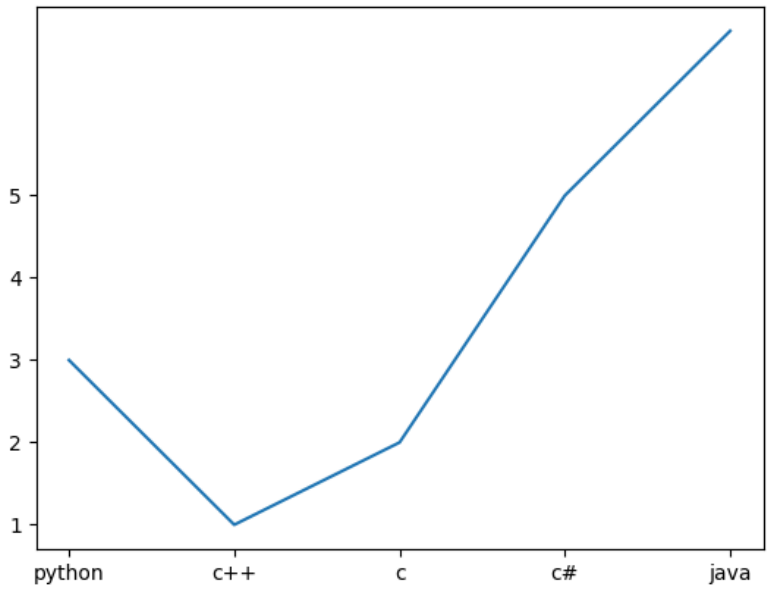
```
x = [1,2,3,4,5]
y = [3,1,2,5,7]

plt.plot(x,y)
```

```
plt.xticks(x)
plt.yticks(x)

plt.show()
```



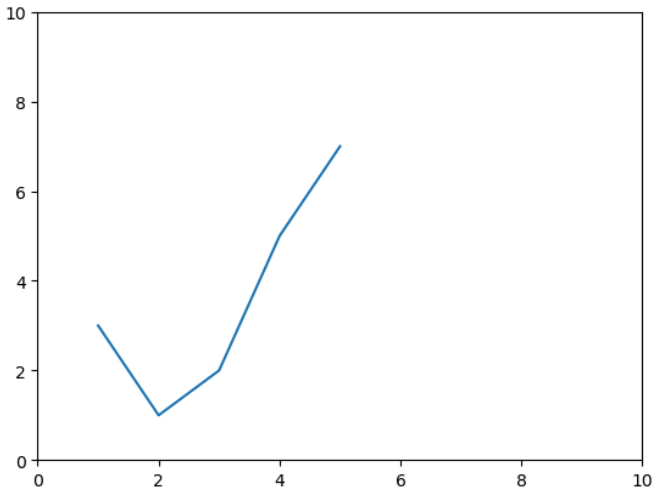
- **Also we can see by labels**

```
x = [1,2,3,4,5]
y = [3,1,2,5,7]

plt.plot(x,y)
plt.xticks(x,labels=["python","c++","c","c#","java"])
plt.yticks(x)
plt.show()
```



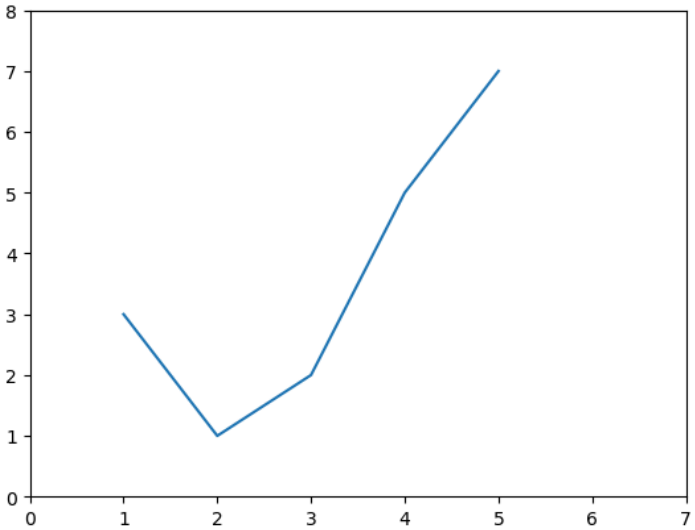
3) Also we can have axis with limits -

```
x = [1,2,3,4,5]
y = [3,1,2,5,7]
plt.plot(x,y)

plt.xlim(0,10)
plt.ylim(0,10)

plt.show()
```



4) And another method - plt.axis([for both x and y])

```
x = [1,2,3,4,5]
y = [3,1,2,5,7]
plt.plot(x,y)
plt.axis([0,7,0,8])
plt.show()
```



Work with text

Working with Text

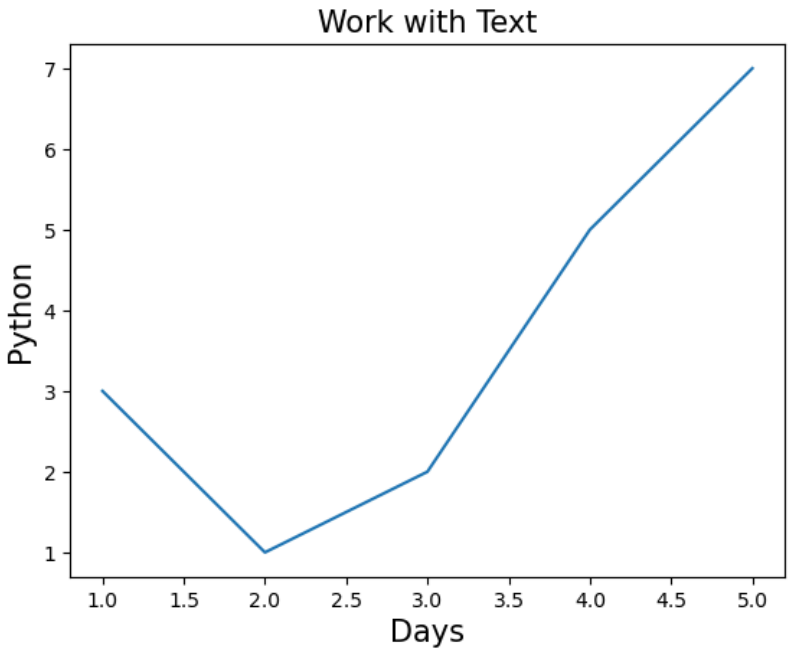
Text in Matplotlib

WsCube Tech

text	Add text at an arbitrary location of the Axes.
annotate	Add an annotation, with an optional arrow, at an arbitrary location of the Axes.
xlabel	Add a label to the Axes's x-axis.
ylabel	Add a label to the Axes's y-axis.
title	Add a title to the Axes.

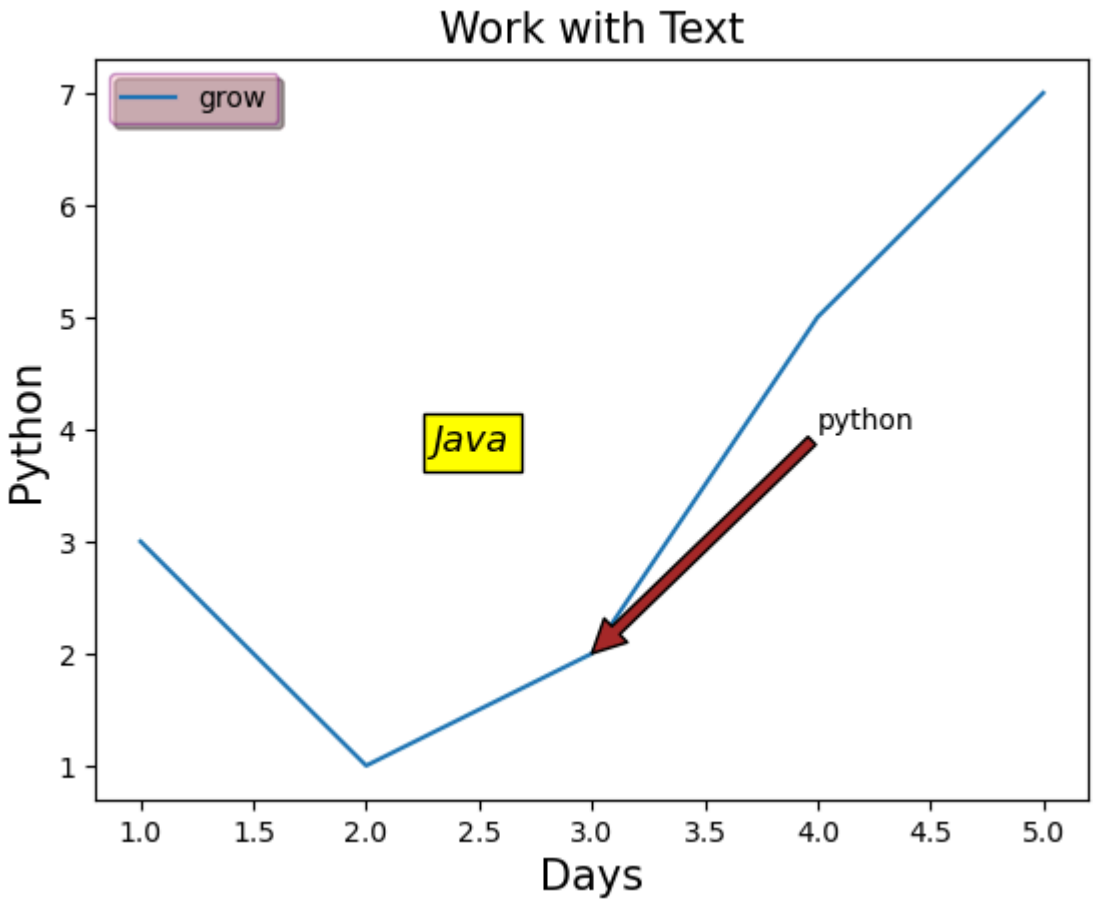
1) Normal

```
x = [1,2,3,4,5]
y = [3,1,2,5,7]

plt.plot(x,y)
plt.title("Work with Text",fontsize=15)
plt.xlabel("Days",fontsize=15)
plt.ylabel("Python",fontsize=15)

plt.show()
```



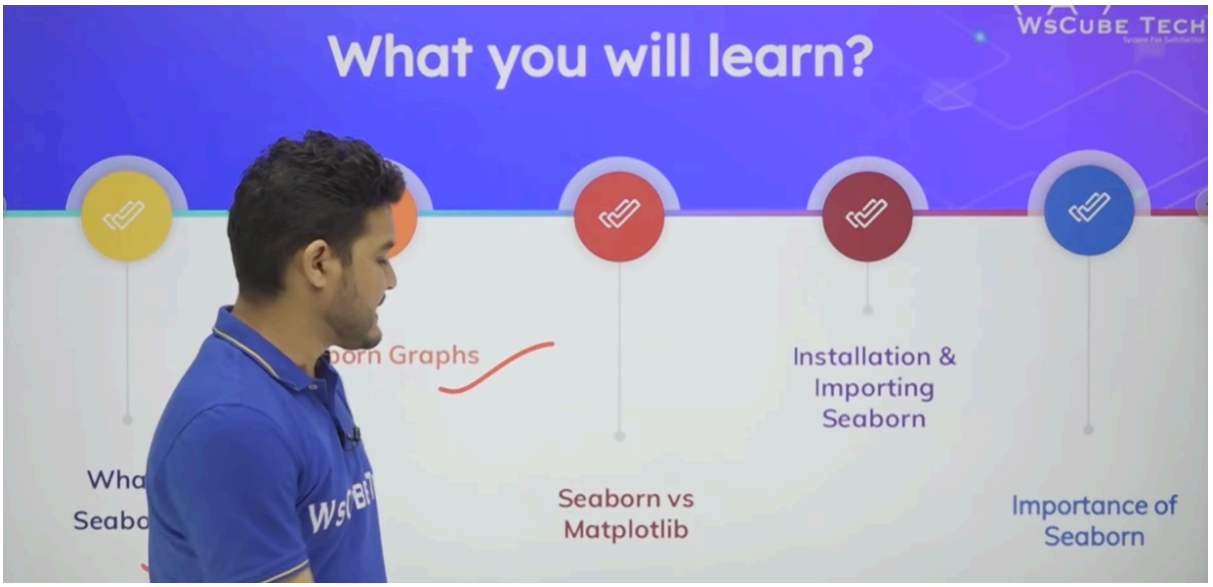
2) Full - text, annotate(arrow), legend

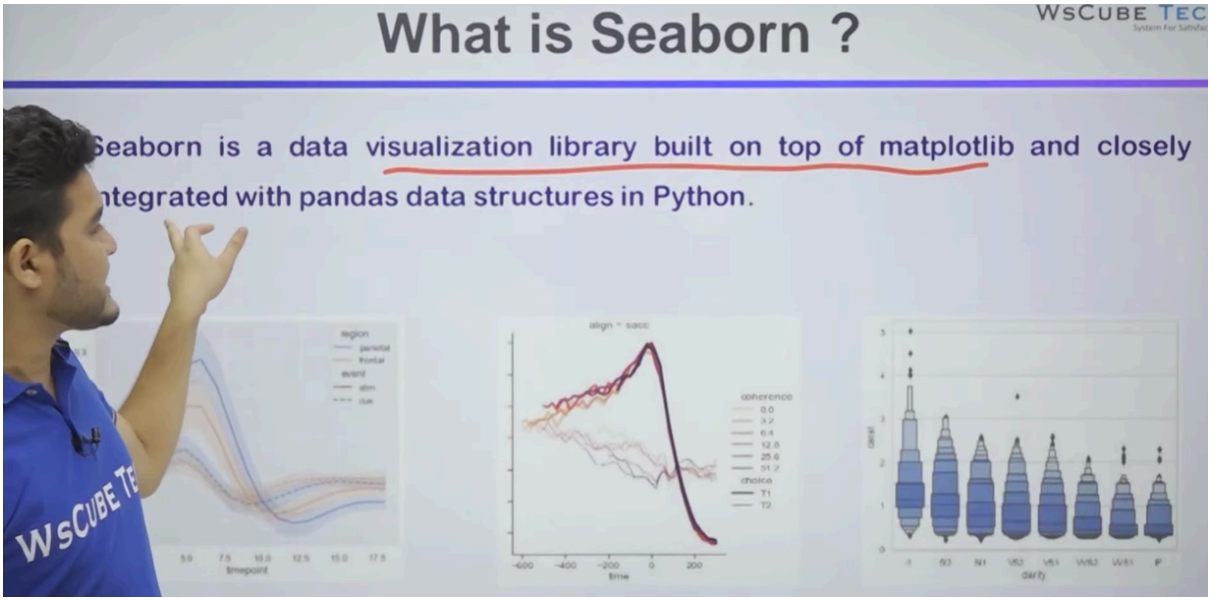
```
x = [1,2,3,4,5]
y = [3,1,2,5,7]

plt.plot(x,y)
plt.title("Work with Text",fontsize=15)
plt.xlabel("Days",fontsize=15)
plt.ylabel("Python",fontsize=15)

plt.text(2.3,3.8,"Java",fontsize=13,style="italic",bbox={"fa
cecolor":"Yellow"})

plt.annotate("python",xy=(3,2),xytext=(4,4),arrowprops=dict(
facecolor="Brown",shrink=100))

plt.legend(["grow"],loc=2,facecolor="Pink",edgecolor="purple
",framealpha=0.4,shadow=True)

plt.show()
```

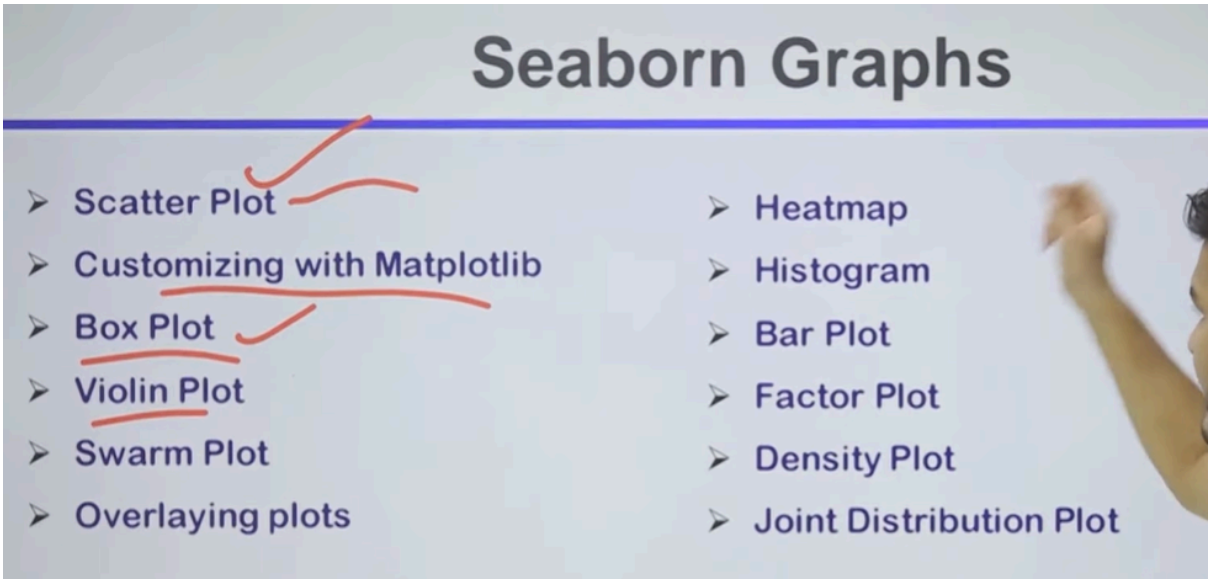


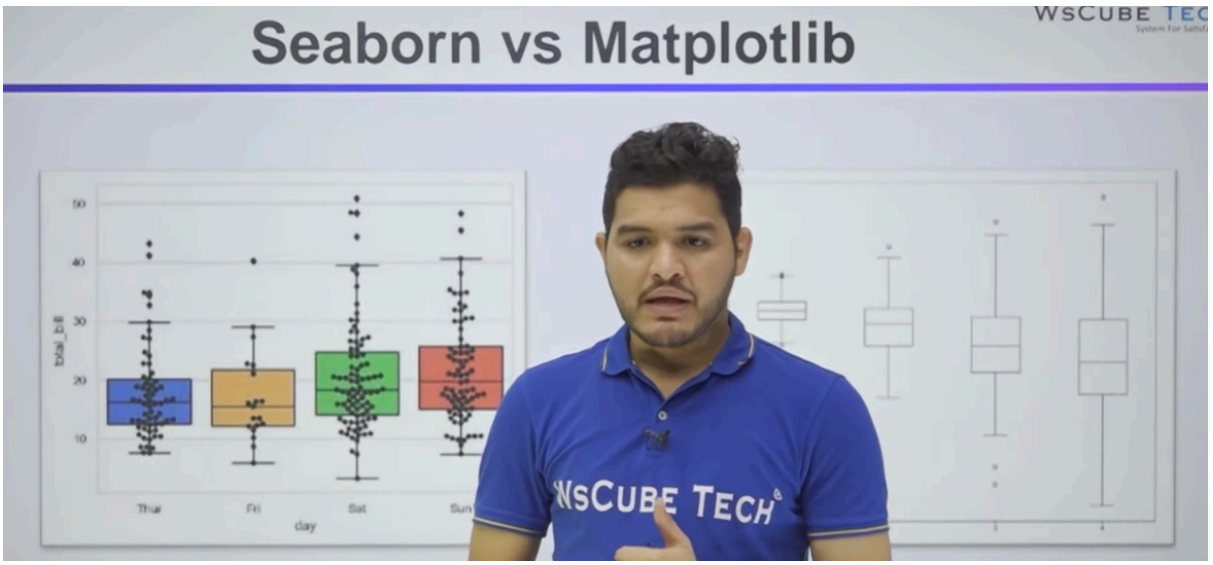
Seaborn

Introduction

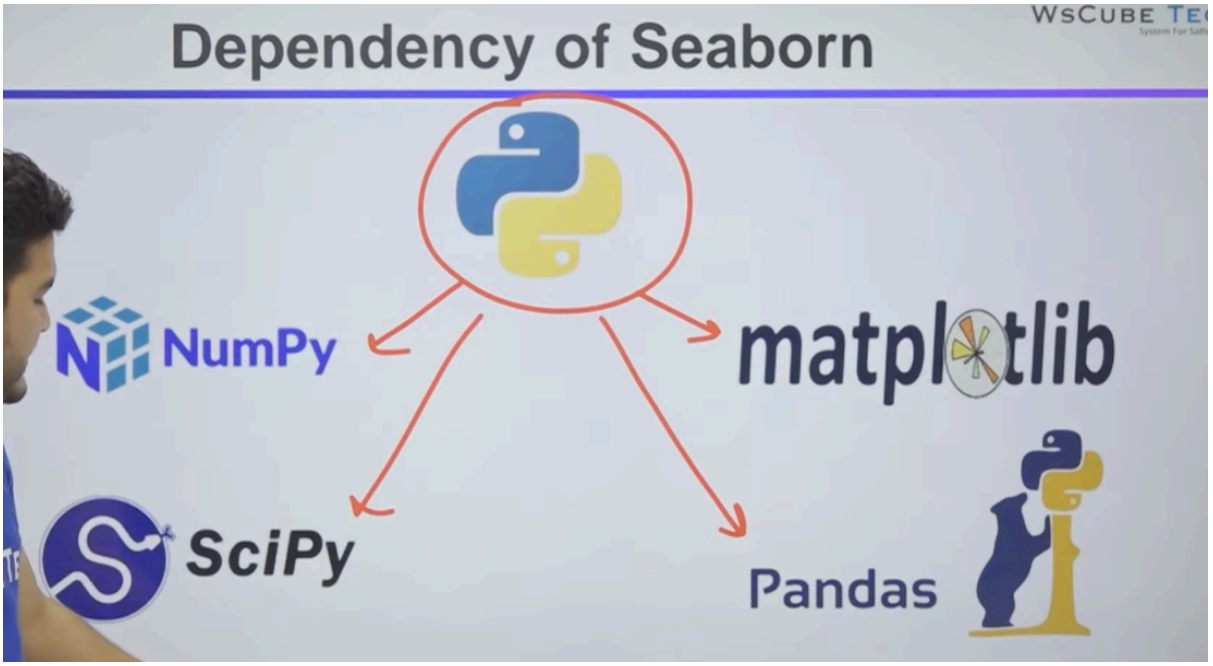


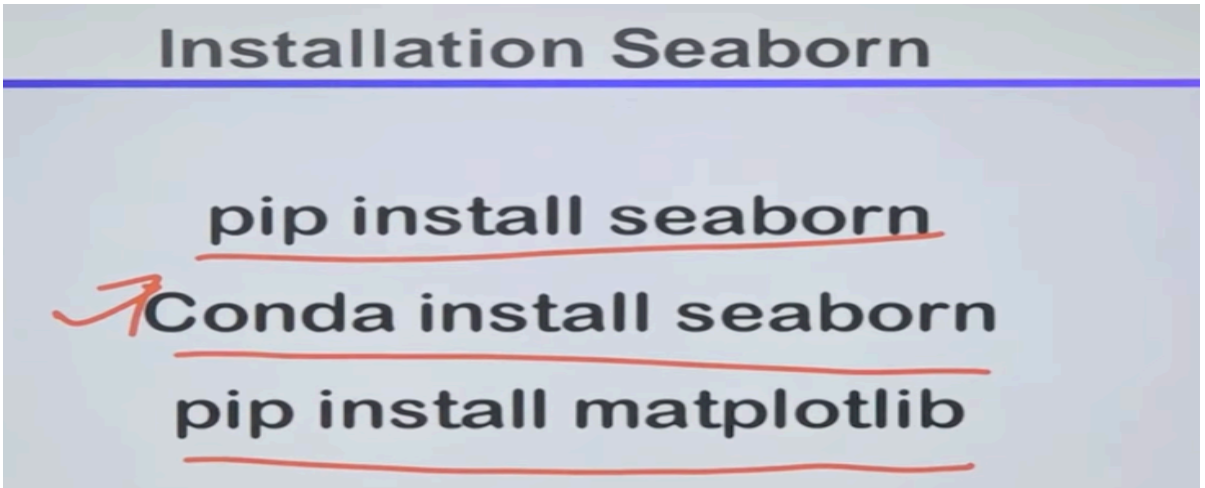

What is Seaborn ?
WsCube Tec
Seaborn is a data visualization library built on top of matplotlib and closely
integrated with pandas data structures in Python.

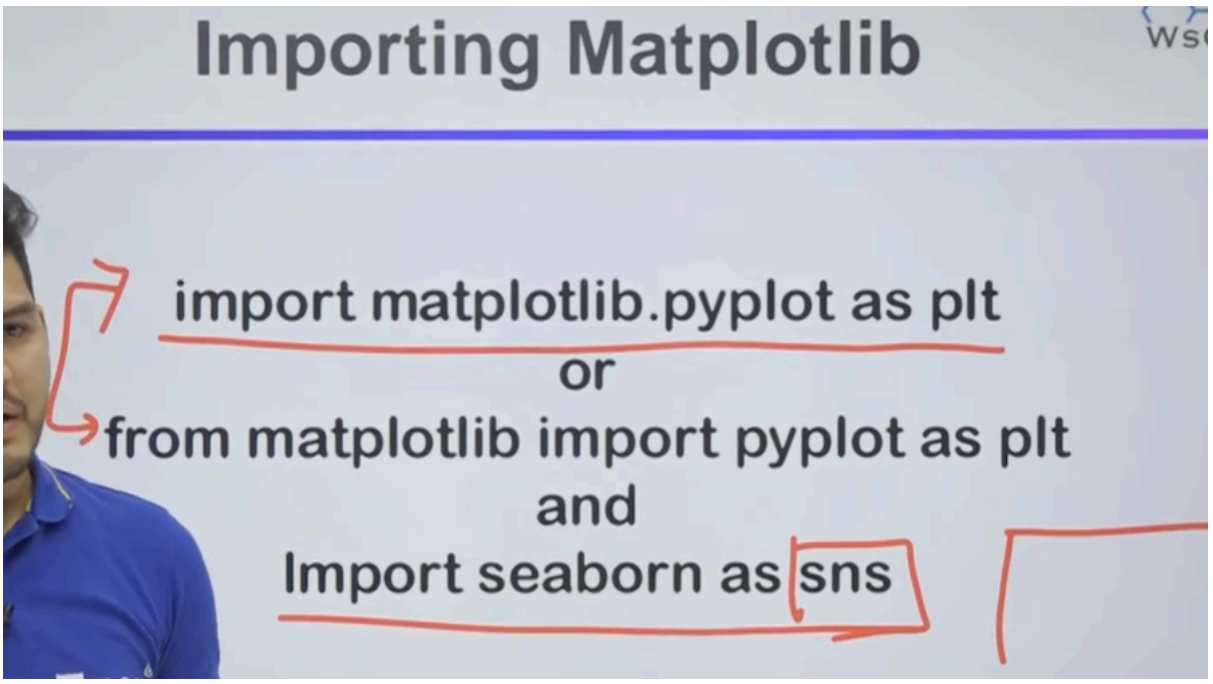




Seaborn - best visualisation and within less code
Matplotlib - less visualisation and if want best visualisation then needs more and hard code





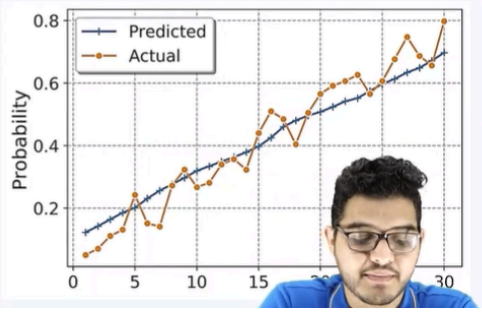

Importing Matplotlib
import matplotlib.pyplot as plt
or
from matplotlib import pyplot as plt
and
Import seaborn as sns

line plot

Line Plot

Seaborn line plot

```
import seaborn as sns
import matplotlib.pyplot as plt

data = sns.load_dataset("  ")
sns.lineplot(x="  ", y="  ", data = data)
plt.show()
```

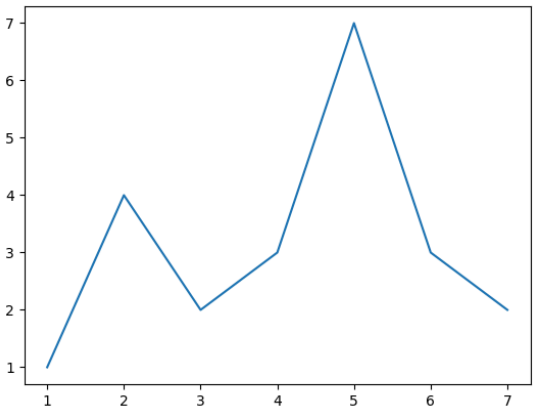


1) Normal

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

- **By matplotlib**

```
var = [1,2,3,4,5,6,7]
var_1 = [1,4,2,3,7,3,2]

#for matplotlib
plt.plot(var,var_1)
plt.show()
```

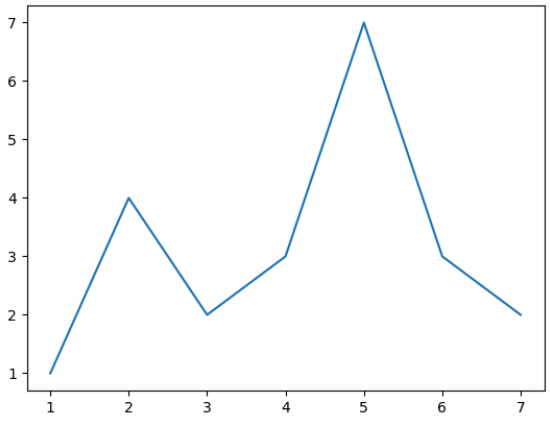


- **By Seaborn**

  a) **Normal method**

```
import seaborn as sns
import matplotlib.pyplot as plt
```

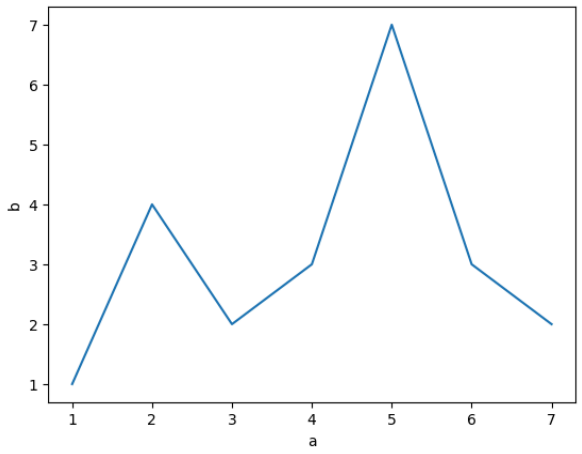
```
import pandas as pd
var = [1,2,3,4,5,6,7]
var_1 = [1,4,2,3,7,3,2]
#for seaborn

sns.lineplot(x=var,y=var_1)
plt.show()
```



here it not define axis.

b) By Define dataframe

```
var = [1,2,3,4,5,6,7]
var_1 = [1,4,2,3,7,3,2]

x = pd.DataFrame({"a":var,"b":var_1})

sns.lineplot(x="a",y="b",data=x)
plt.show()
```



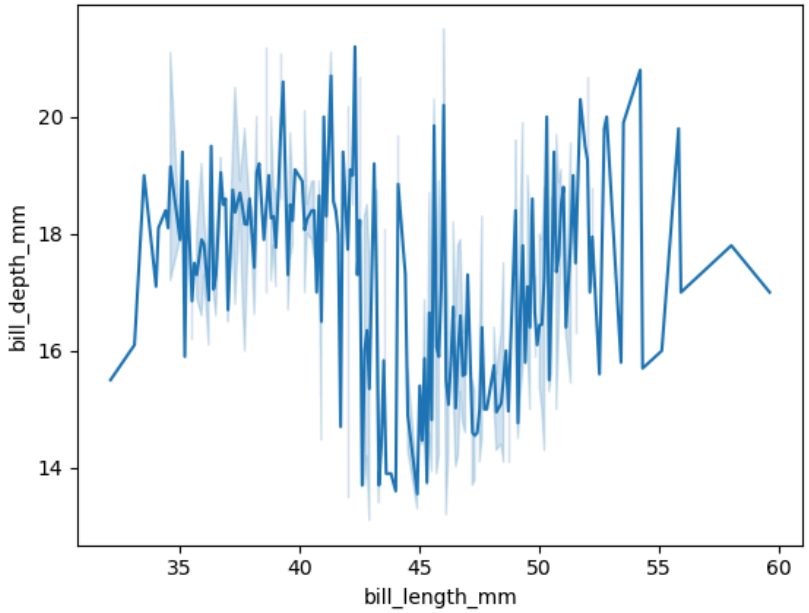
Define axis...

2) Normal but on big data -

```
dt = sns.load_dataset("penguins")
Dt #--data loaded
```

```
sns.lineplot(x="bill_length_mm",y="bill_depth_mm",data=dt)
```

```
plt.show()
```



3) Modify with big data

Hue - reference visualisation about what

Style - at what you want style (sex) – one is straing 2nd is dotted line

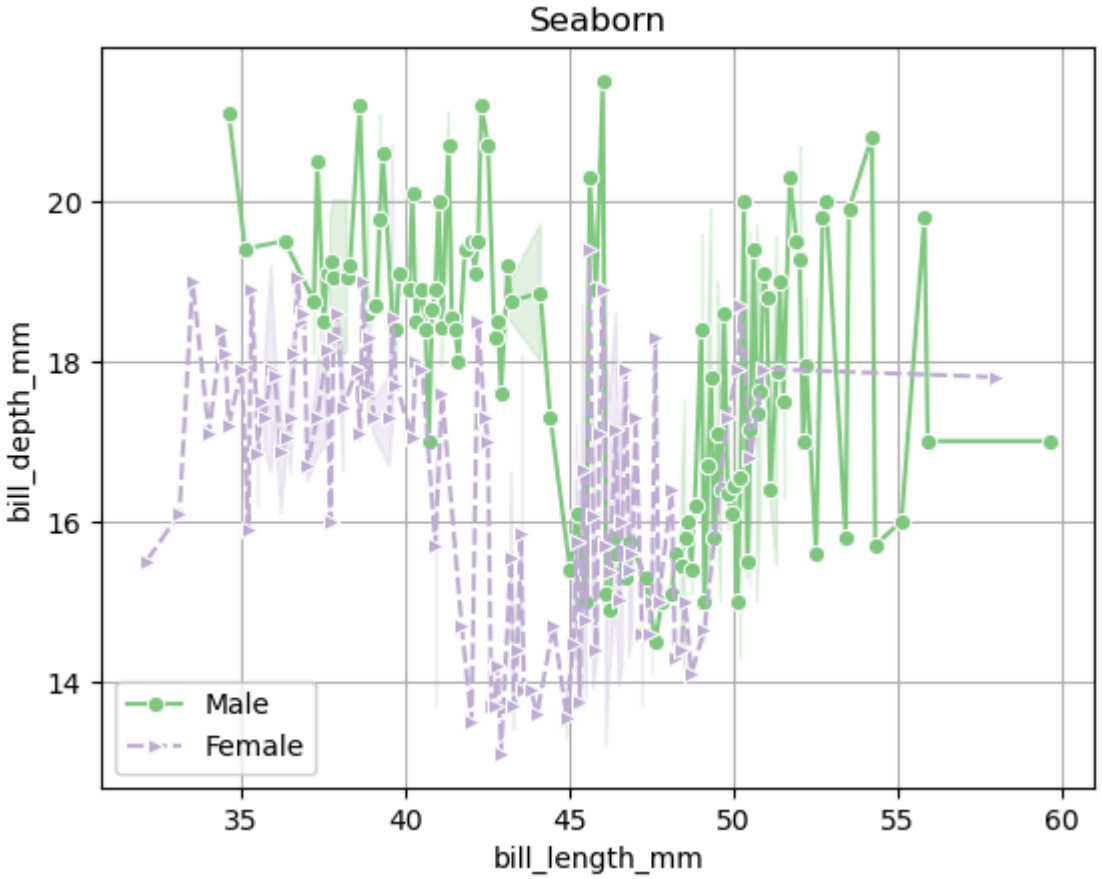
Palette - colour combinations.. Given below

```
ValueError: 'A' is not a valid value for name; supported values are 'Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r',
'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys',
'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2',
'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_
r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1',
'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'Yl
Gn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'b
one_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_
r', 'crest', 'crest_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'flare', 'flare_r', 'gist_earth', 'gist_g
ray', 'gist_gray_r', 'gist_heat', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r',
ist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r',
'hsv', 'hsv_r', 'icefire', 'icefire_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'mako',
ctral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r',
r', 'rocket', 'rocket_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab10
20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight',
ght_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'vlag', 'vlag_r', 'winter', 'winter_r'

During handling of the above exception, another exception occurred:
```

Markers - point or coordinate style

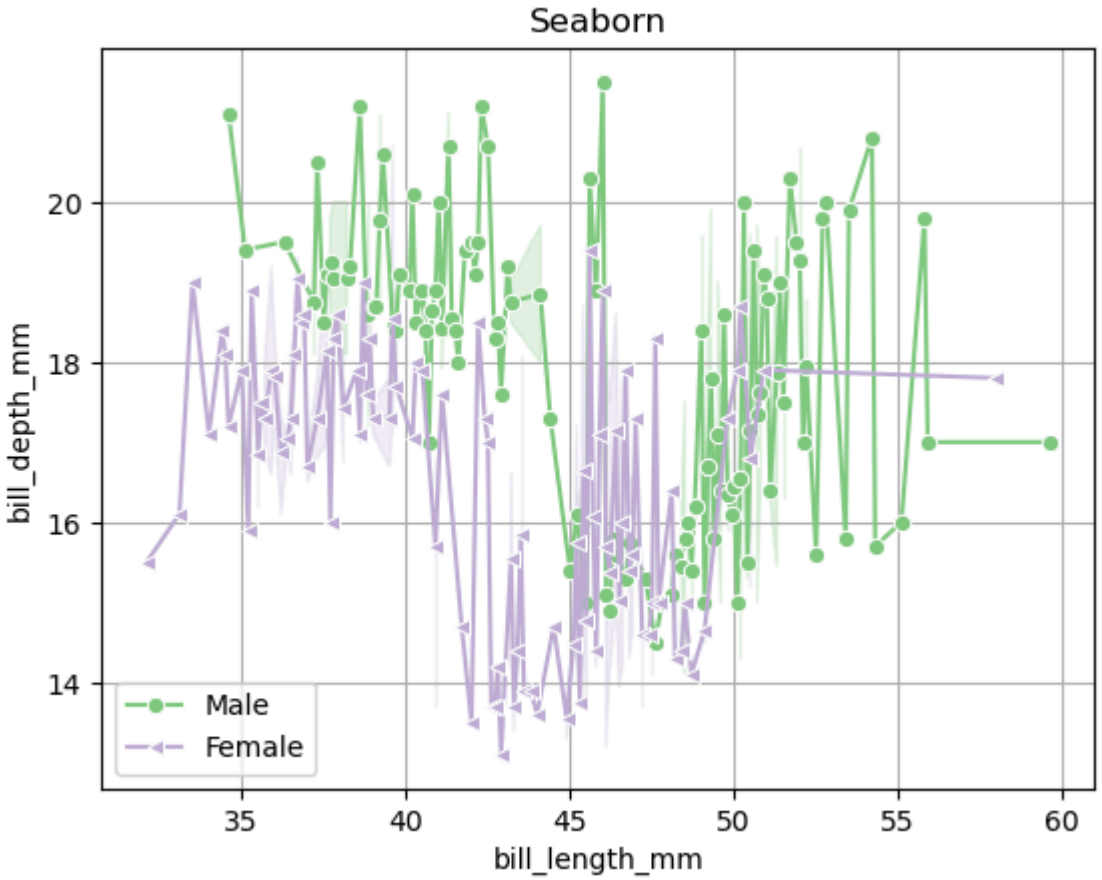
```
sns.lineplot(x="bill_length_mm",y="bill_depth_mm",data=dt,hu
e="sex",style="sex",palette="Accent",markers=["o",">"],legen
d="full")
```

```
plt.title("Seaborn")
plt.grid()
plt.legend(loc=3)
plt.show()
```



- **Dashes - (remove style)-- doted line convert in simple line**

```
sns.lineplot(x="bill_length_mm",y="bill_depth_mm",data=dt,hue="sex",style="sex",palette="Accent",markers=["o","<"],dashes=False,legend="full")

plt.title("Seaborn")
plt.grid()
plt.legend(loc=3)
plt.show()
```


Seaborn
bill_depth_mm
bill_length_mm
20
18
16
14
35
40
45
50
55
60
Male
Female

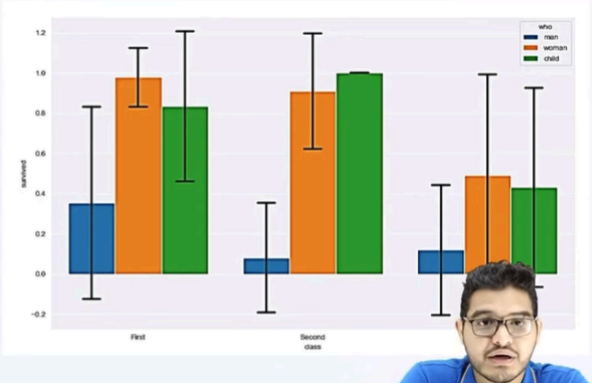
bar plot

Bar Plot

Barplot Python Seaborn

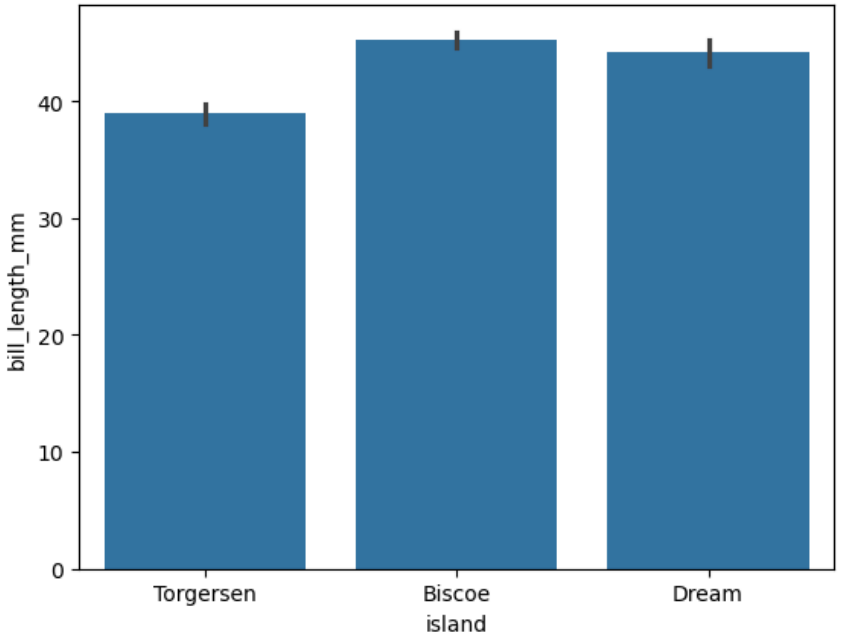
WsCube Tech

```
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset(' ')
sns.barplot(x = ' ', y = ' ', data = df)
plt.show()
```



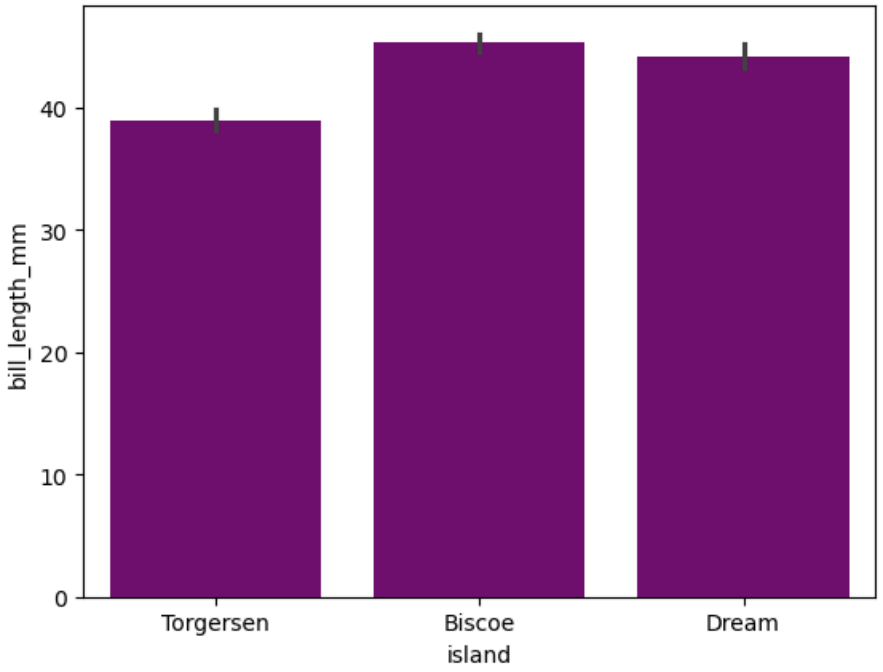
1) Normal

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

dt = sns.load_dataset("penguins")
dt

sns.barplot(x=dt.island,y=dt.bill_length_mm)
plt.show()
```



- **Another way with colour**

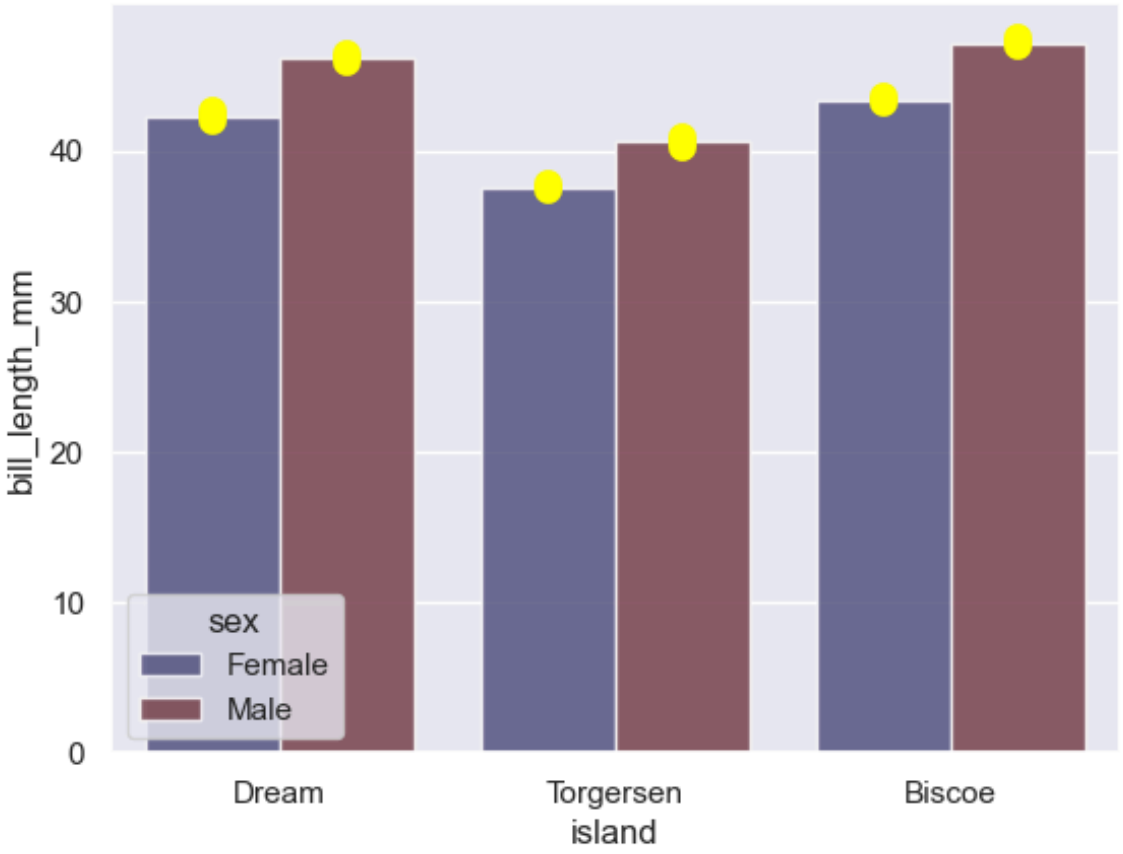
```
sns.barplot(x="island",y="bill_length_mm",data=dt,color="purple")
plt.show()
```



2) Modified

Ci - controls *what uncertainty is shown*
N_boot - controls *how accurately that uncertainty is estimated*.
Errcolor - colour of err (above little line)
Errwidth - width of of err
sns.set(style="darkgrid") - grid in seaborn
Order, orient, alpha, hue, palette (same)

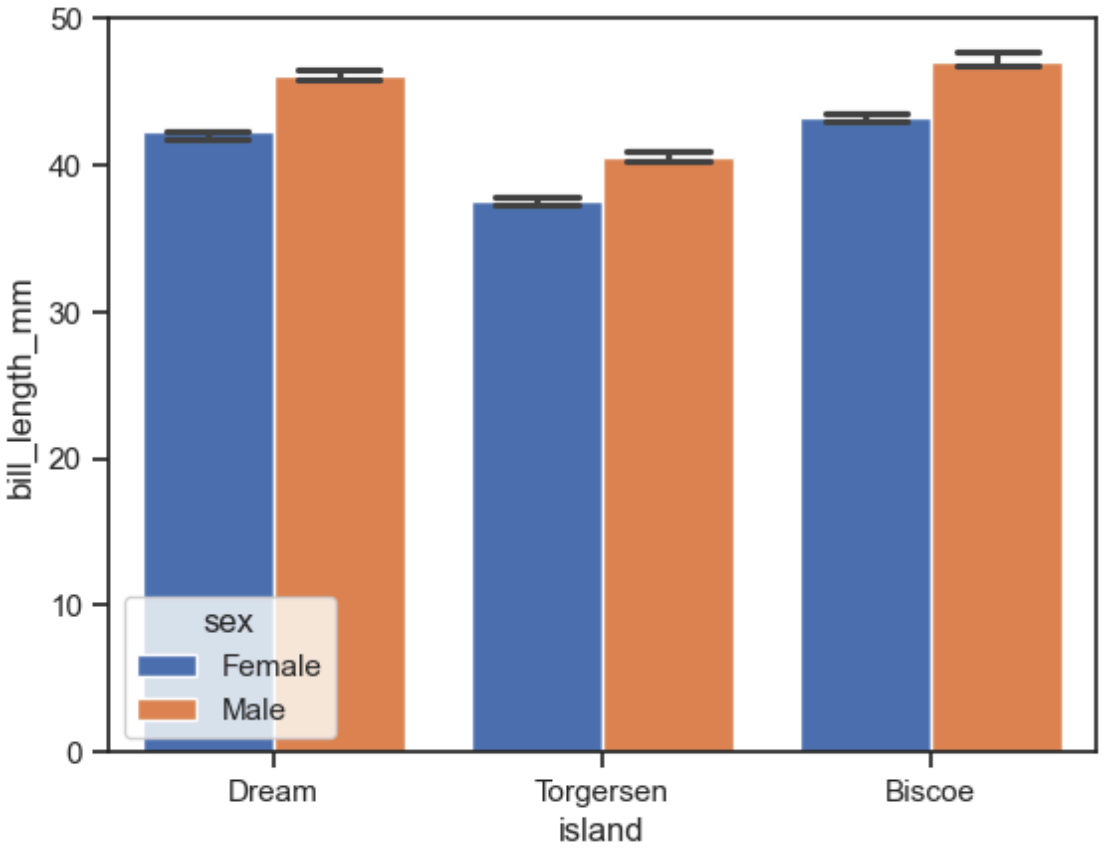
```
sns.set(style="darkgrid")

order1 = ["Dream","Torgersen","Biscoe"]

sns.barplot(x="island",y="bill_length_mm",data=dt,hue="sex",order=order1,hue_order=["Female","Male"],ci=40,n_boot=50,orient="v",palette="icefire",errcolor="yellow",errwidth=10,alpha=0.8)
plt.show()
```



3) Modified (2)

Saturation - it contrast colour low or high
capsize - it change size of cap

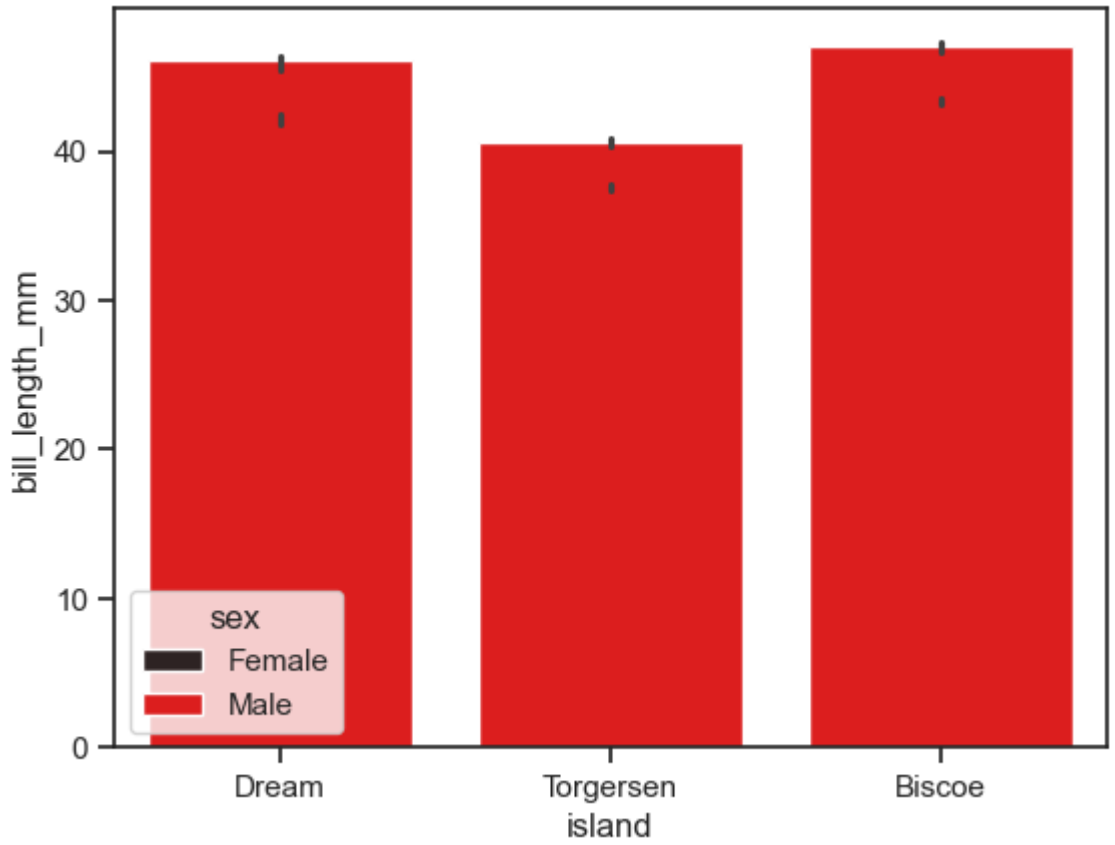
```
sns.set(style="ticks")
order1 = ["Dream","Torgersen","Biscoe"]

sns.barplot(x="island",y="bill_length_mm",data=dt,hue="sex",
order=order1,hue_order=["Female","Male"],ci=40,n_boot=50,ori
ent="v",saturation=1,capsize=0.5)

plt.show()
```



4) Modified (3)

Dodge - it combine references which separate graphs was managing by hue

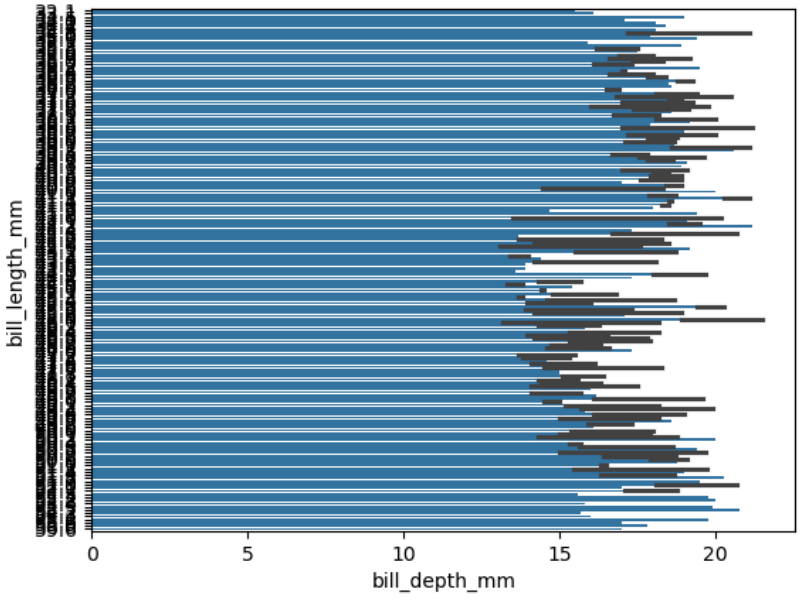
```
order1 = ["Dream","Torgersen","Biscoe"]

sns.barplot(x="island",y="bill_length_mm",data=dt,hue="sex",
order=order1,hue_order=["Female","Male"],ci=40,n_boot=50,ori
ent="v",color="red",dodge=False)

plt.show()
```



5) Horizontal

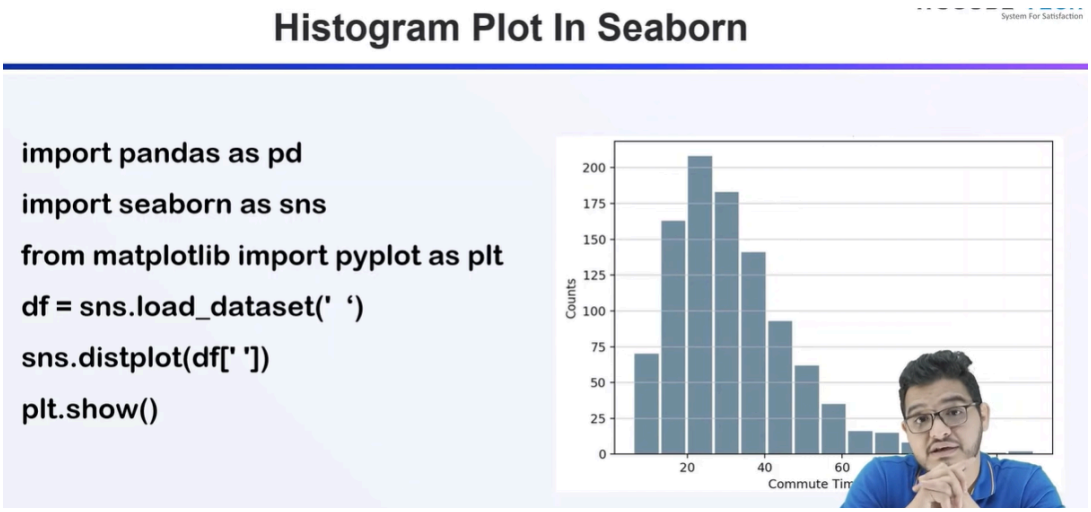
- **Condition -> both or all columns should be numeric**

```
sns.barplot(x="bill_depth_mm",y="bill_length_mm",data=dt,orient="h")
plt.show()
```



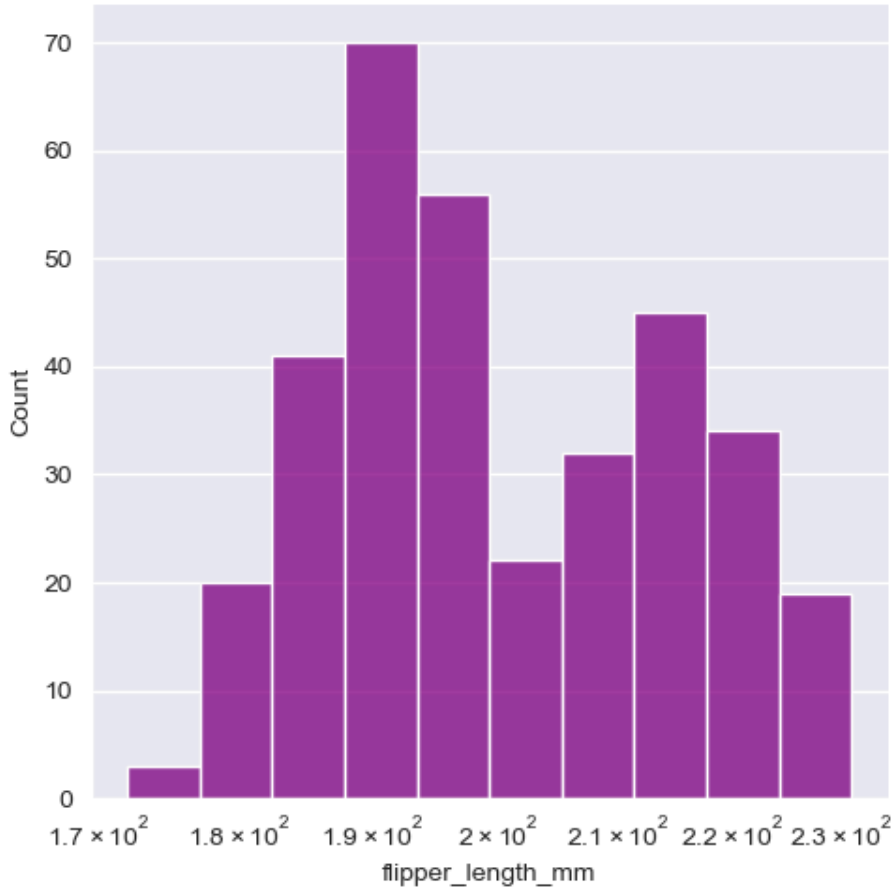
histogram plot

Histogram Plot



1) Normal with log

```
sns.displot(dt["flipper_length_mm"],color="purple",log_scale
=True)

sns.set(style="darkgrid")
plt.show()
```



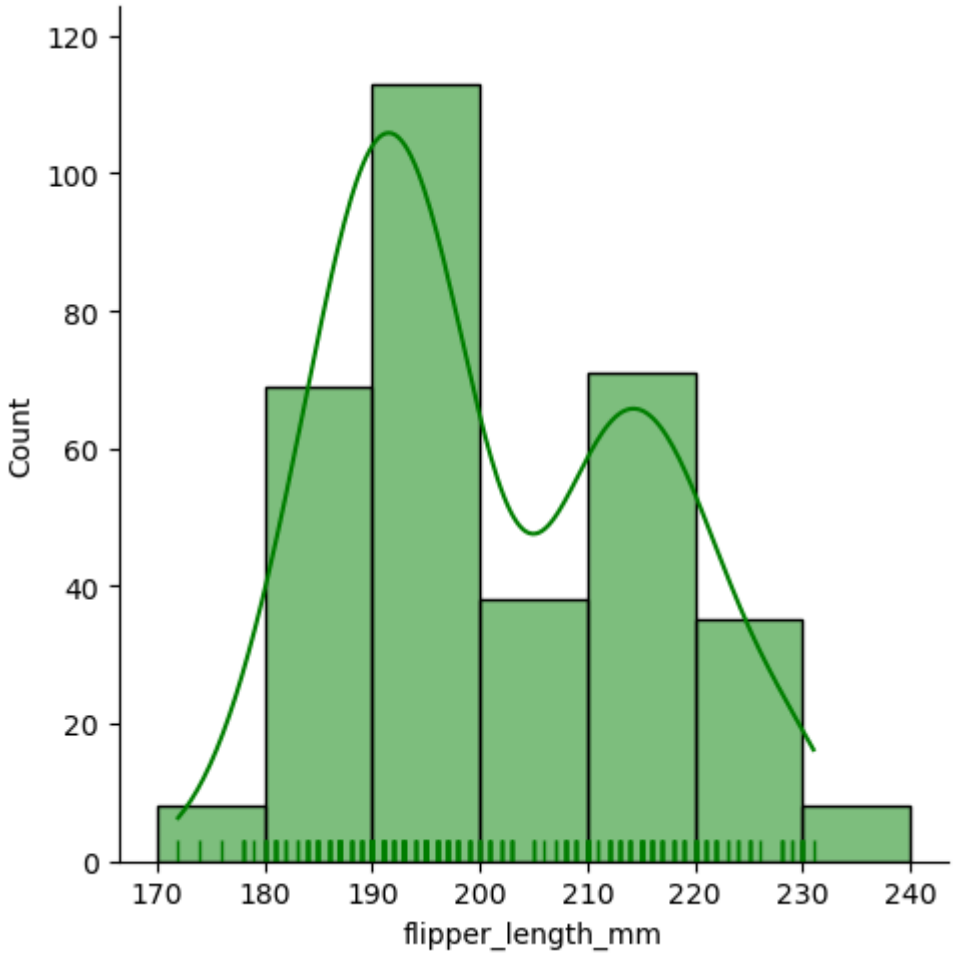
2) Modified

Bins - used to maintain data in fixed interval

Kde (Kernel Density Estimate) - it form line graph also in flexible way

Rug - its make bar graph type very tiny below

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
dt = sns.load_dataset("penguins")
Dt

sns.displot(dt["flipper_length_mm"],bins=[170,180,190,200,210,220,230,240],kde=True,rug=True,color="g")

plt.show()
```



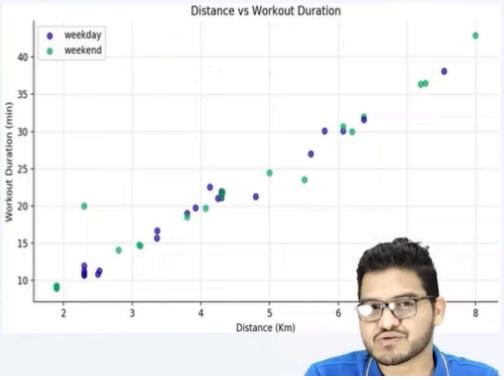
Scatter plot

Scatter Plot

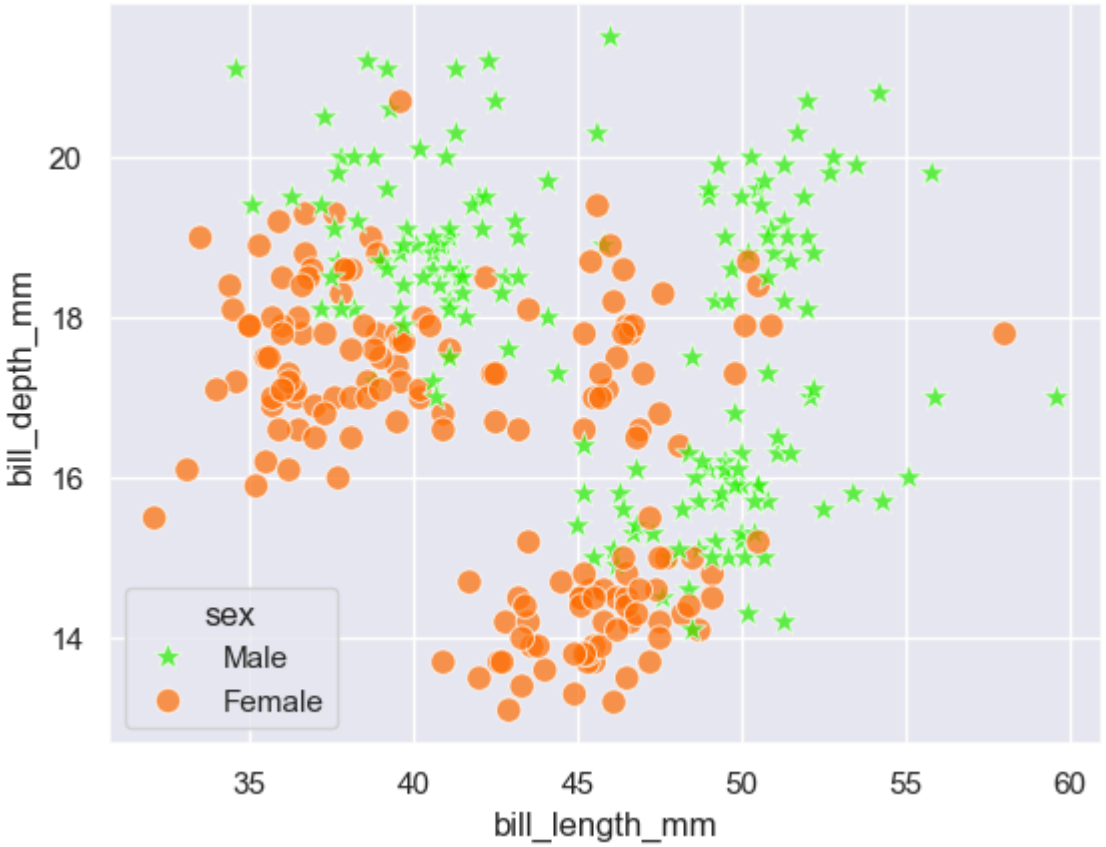
Seaborn Scatter Plot

WSCUBE TECH
System For Satisfaction

```
import seaborn as sns
import matplotlib.pyplot as plt

data = sns.load_dataset("  ")
sns.scatterplot(x=" ", y="  ",
                data=data)
plt.show()
```

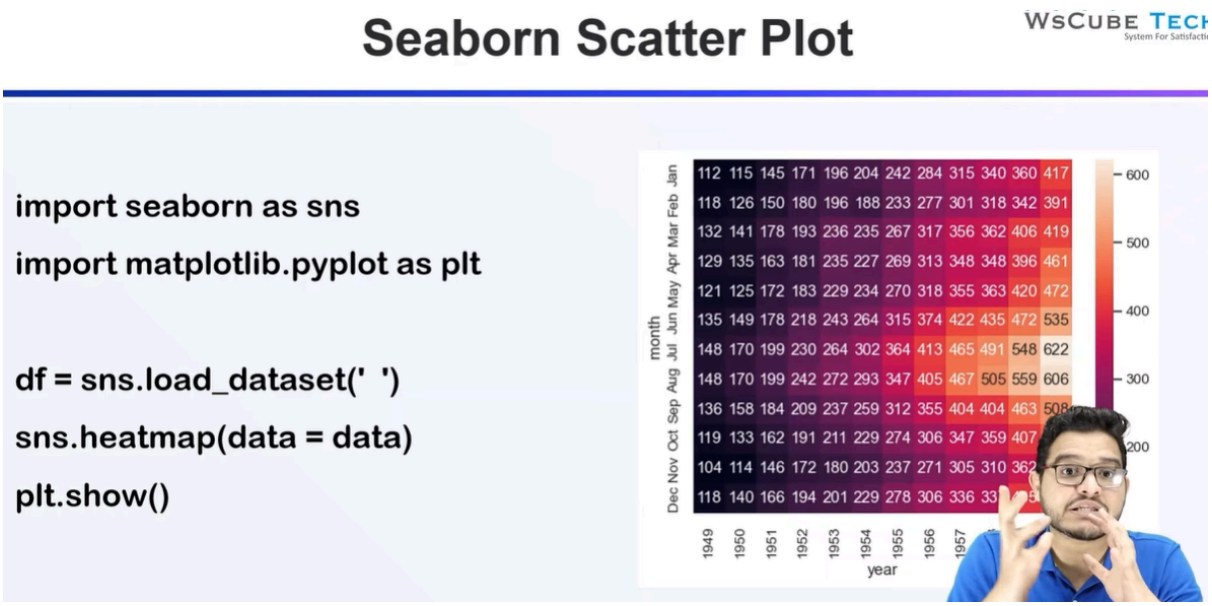



```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

dt = sns.load_dataset("penguins")
dt

m = {"Male":"*","Female":"o"}

sns.scatterplot(x="bill_length_mm",y="bill_depth_mm",data=dt,hue=
"sex",style="sex",size="sex",sizes=(80,110),palette="gist_ncar",a
lpha=0.7,markers=m)

plt.show()
```


20
18
16
14
bill_depth_mm
35
40
45
50
55
60
bill_length_mm
sex
Male
Female

Heatmap plot

Heatmap Plot

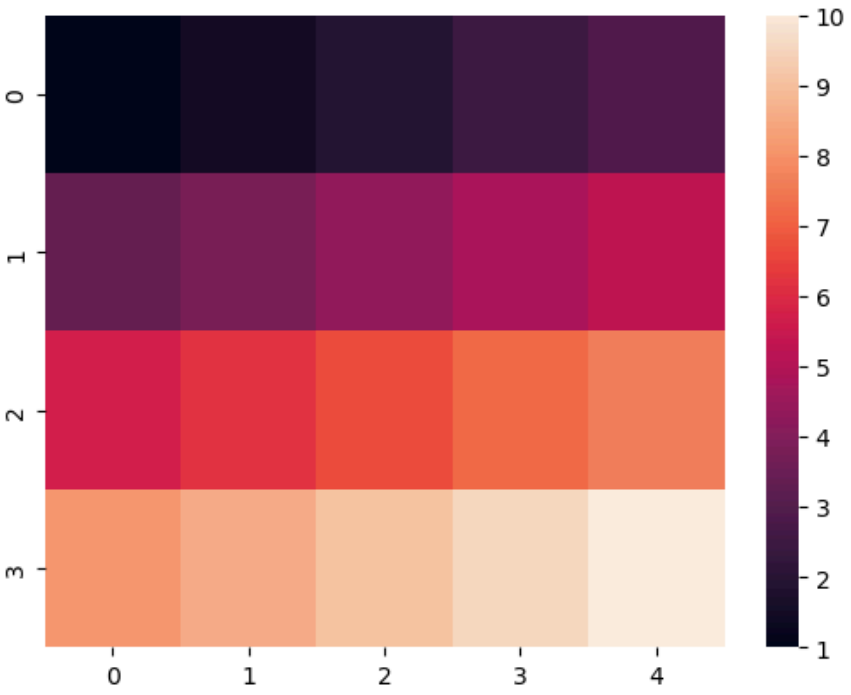


1) Normal

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

var = np.linspace(1,10,20).reshape(4,5)
var
```

```
array([[ 1.        ,  1.47368421,  1.94736842,  2.42105263,  2.89473684],
       [ 3.36842105,  3.84210526,  4.31578947,  4.78947368,  5.26315789],
       [ 5.73684211,  6.21052632,  6.68421053,  7.15789474,  7.63157895],
       [ 8.10526316,  8.57894737,  9.05263158,  9.52631579, 10.        ]])
```

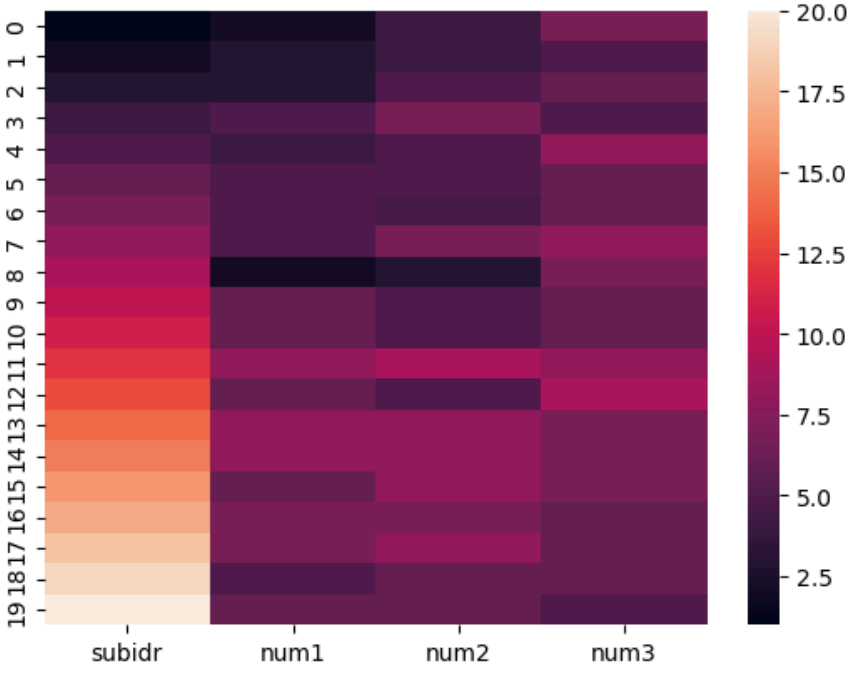
```
sns.heatmap(var)
plt.show()
```



- **By Large dataset**

```
data = sns.load_dataset("anagrams")
dt = data.drop(columns=["attnr"],axis=1)
Dt
```

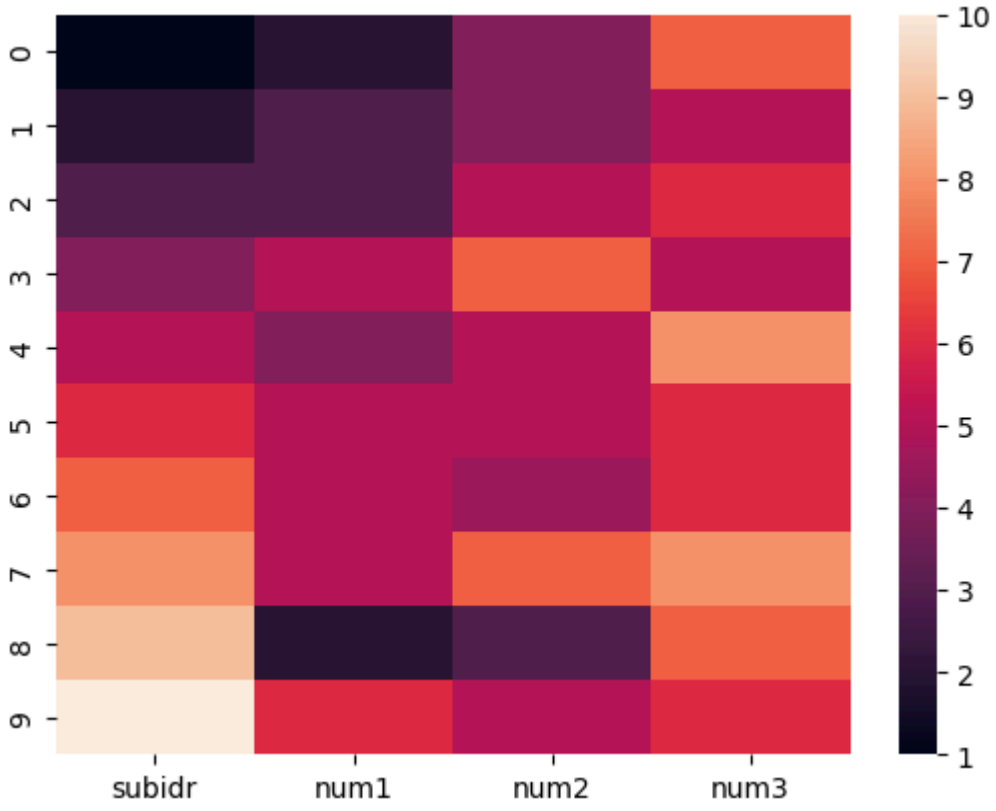
```
sns.heatmap(dt)
```



```
x = dt.head(10)
x
```

	subidr	num1	num2	num3
0	1	2	4.0	7
1	2	3	4.0	5
2	3	3	5.0	6
3	4	5	7.0	5
4	5	4	5.0	8
5	6	5	5.0	6
6	7	5	4.5	6
7	8	5	7.0	8
8	9	2	3.0	7
9	10	6	5.0	6

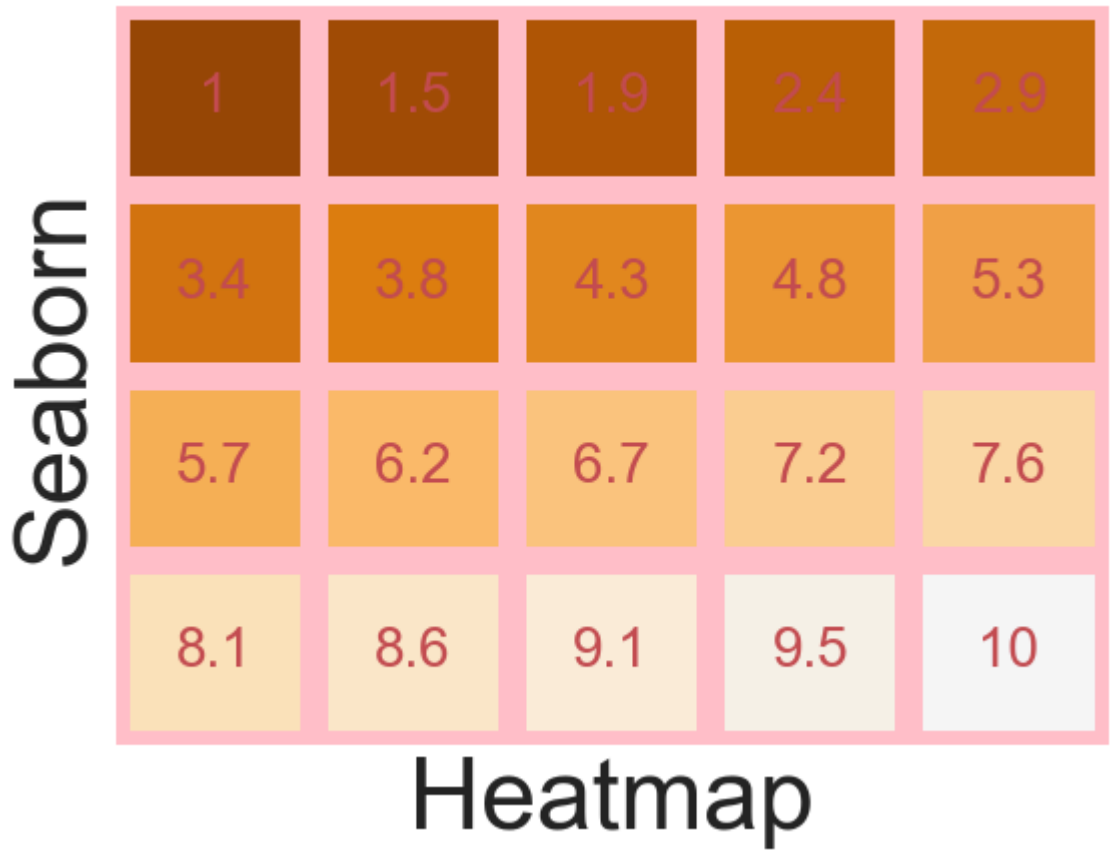
```
sns.heatmap(x)
```



2) Perimeter

Vmin - min point and, **Vmax -** max point
Cmap - colour of heatmap (variations available.. Check google)
Annot - value of arrays and, **Annot_kws -** perimeters of annot
Linewidths - width b/w two box and, **Linecolor -** colour b/w box
Cbar - adjacent bar known as cmap

```
y = {"fontsize":20,"color":"r"}

v =
sns.heatmap(var,vmin=0,vmax=20,cmap="PuOr",annot=True,annot_
kws=y,linewidths=10,linecolor="pink",cbar=False,xticklabels=
False,yticklabels=False)

v.set(xlabel="Heatmap",ylabel="Seaborn")
sns.set(font_scale=2)
plt.show()
```



3) Annot

And if you want to keep annot with some str -

```
ar = np.array([["a1","a2","a3","a4","a5"],
               ["b1","b2","b3","b4","b5"]])
ar
```

✓ 0.0s

```
array([['a1', 'a2', 'a3', 'a4', 'a5'],
       ['b1', 'b2', 'b3', 'b4', 'b5']], dtype='<U2')
```

```
temp_dt = np.linspace(1,8,10).reshape(2,5)
temp_dt
```

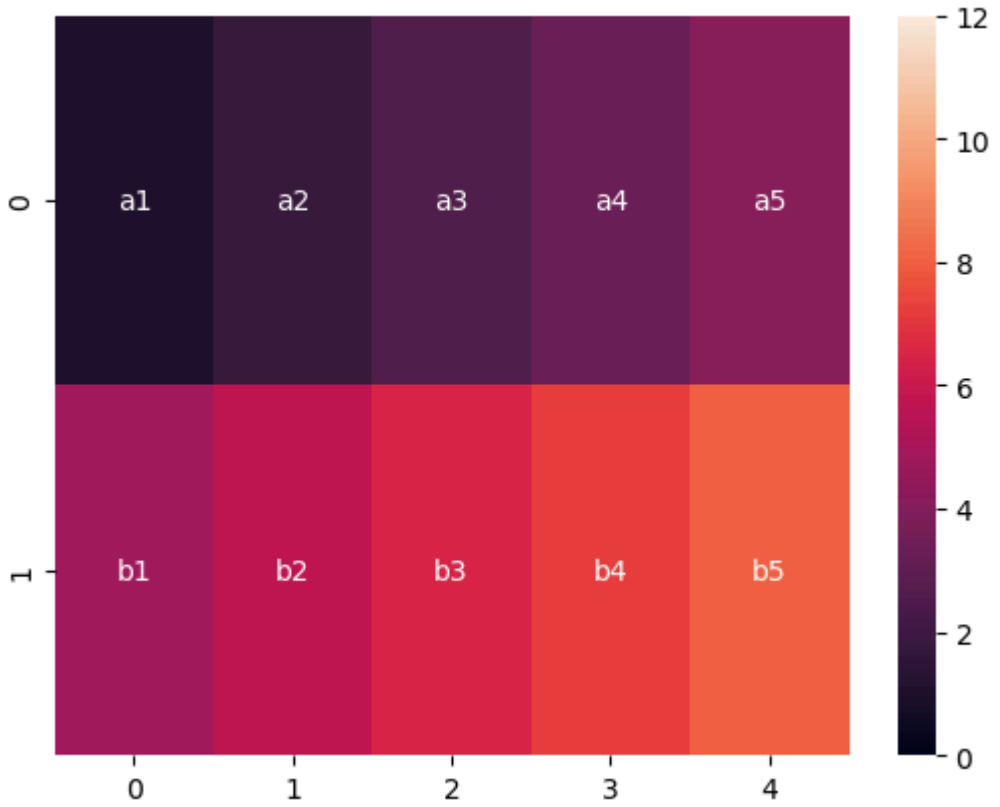
✓ 0.0s

```
array([[1.        , 1.77777778, 2.55555556, 3.33333333, 4.11111111],
       [4.88888889, 5.66666667, 6.44444444, 7.22222222, 8.        ]])
```

```
sns.heatmap(temp_dt,vmin=0,vmax=12,annot=ar,fmt="s")
```

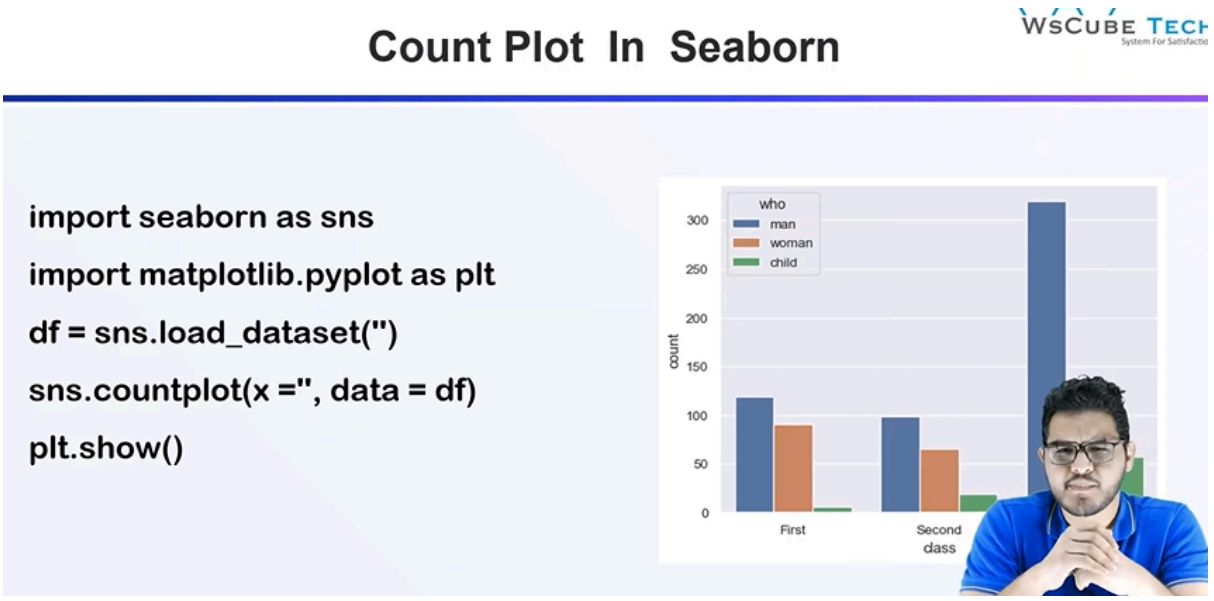
✓ 0.2s

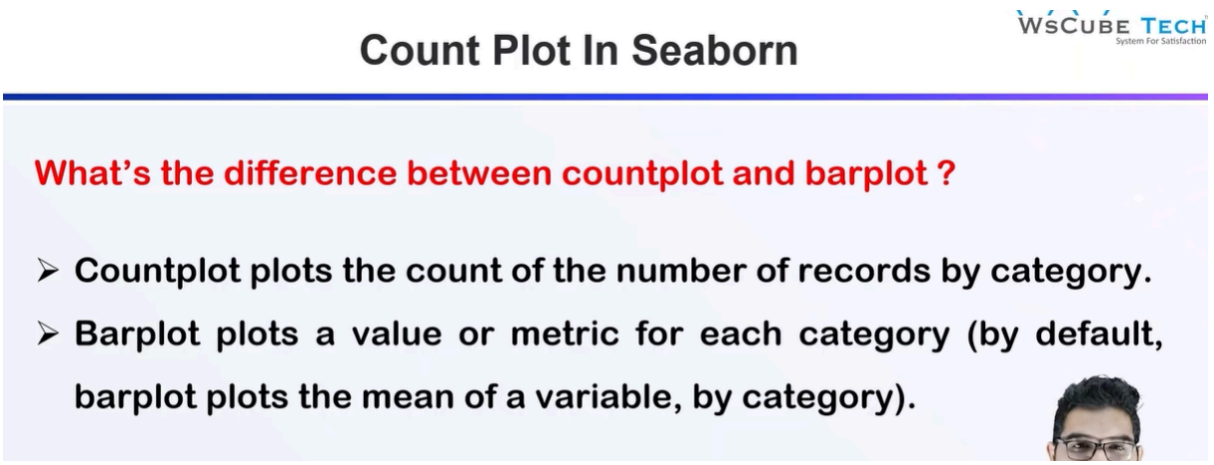
`sns.heatmap(temp_dt,vmin=0,vmax=12,annot=ar,fmt="s")`



count plot

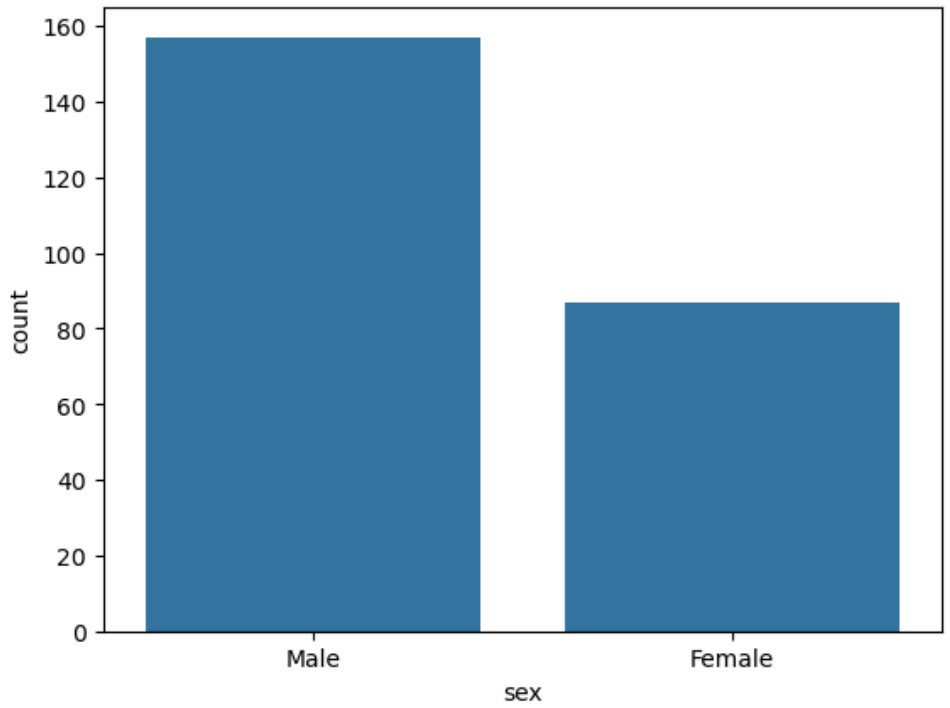
Count Plot




Count Plot In Seaborn
WsCube Tech
What's the difference between countplot and barplot ?
Countplot plots the count of the number of records by category.
Barplot plots a value or metric for each category (by default, barplot plots the mean of a variable, by category).

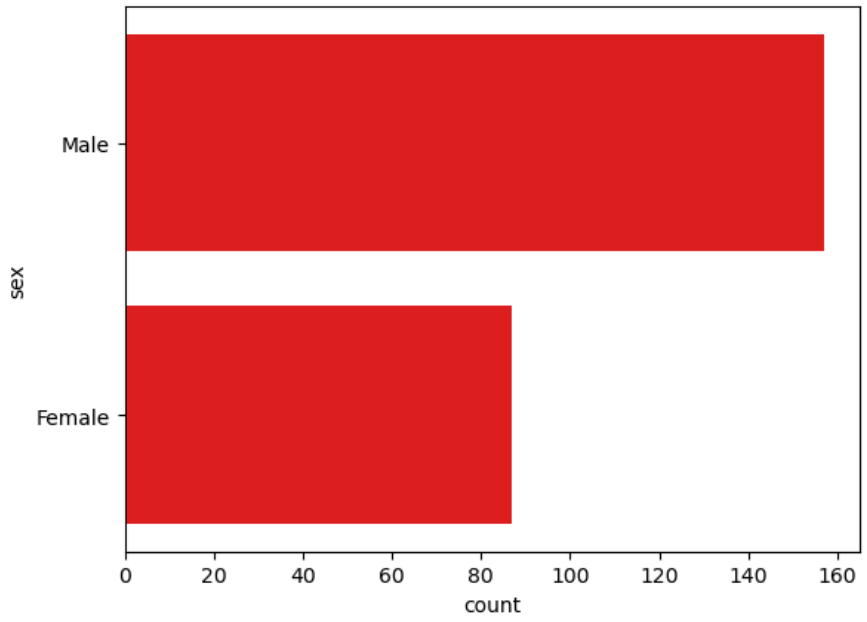
1) Normal

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

var = sns.load_dataset("tips")
var

sns.countplot(x="sex",data=var)
plt.show()
```



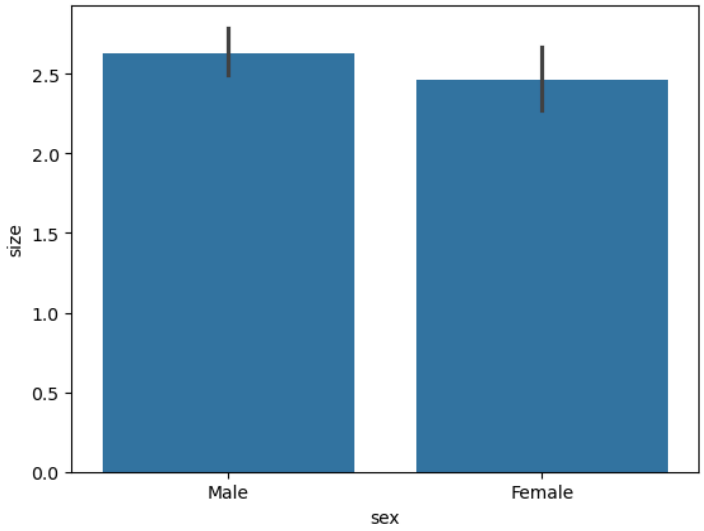
- **Change axis (horizontally)**

```
sns.countplot(y="sex",data=var,color="r")
plt.show()
```



2) And difference b/w countplot and barplot

- **cause barplot shows average**

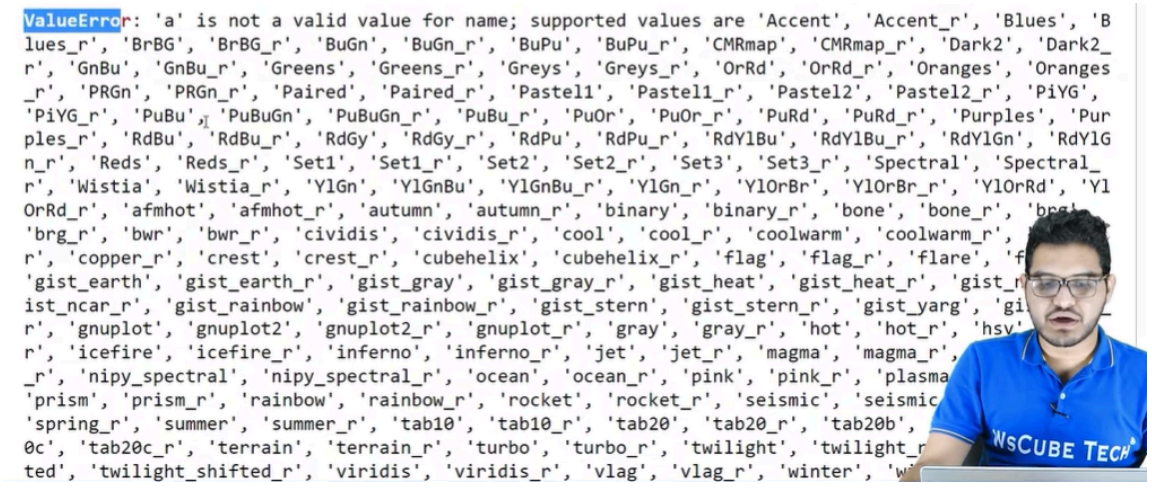


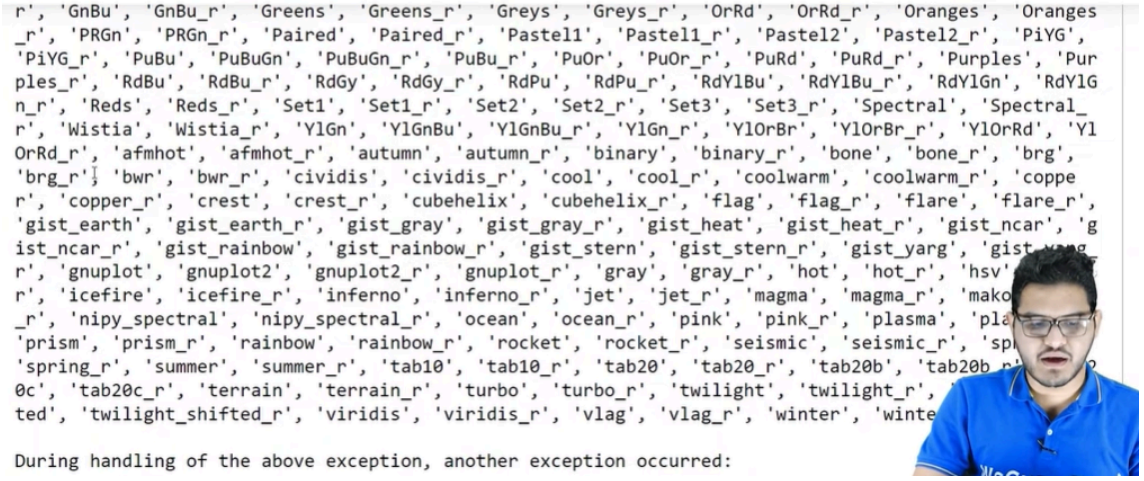
3) Perimeters -

X and y = horizontally and vertically

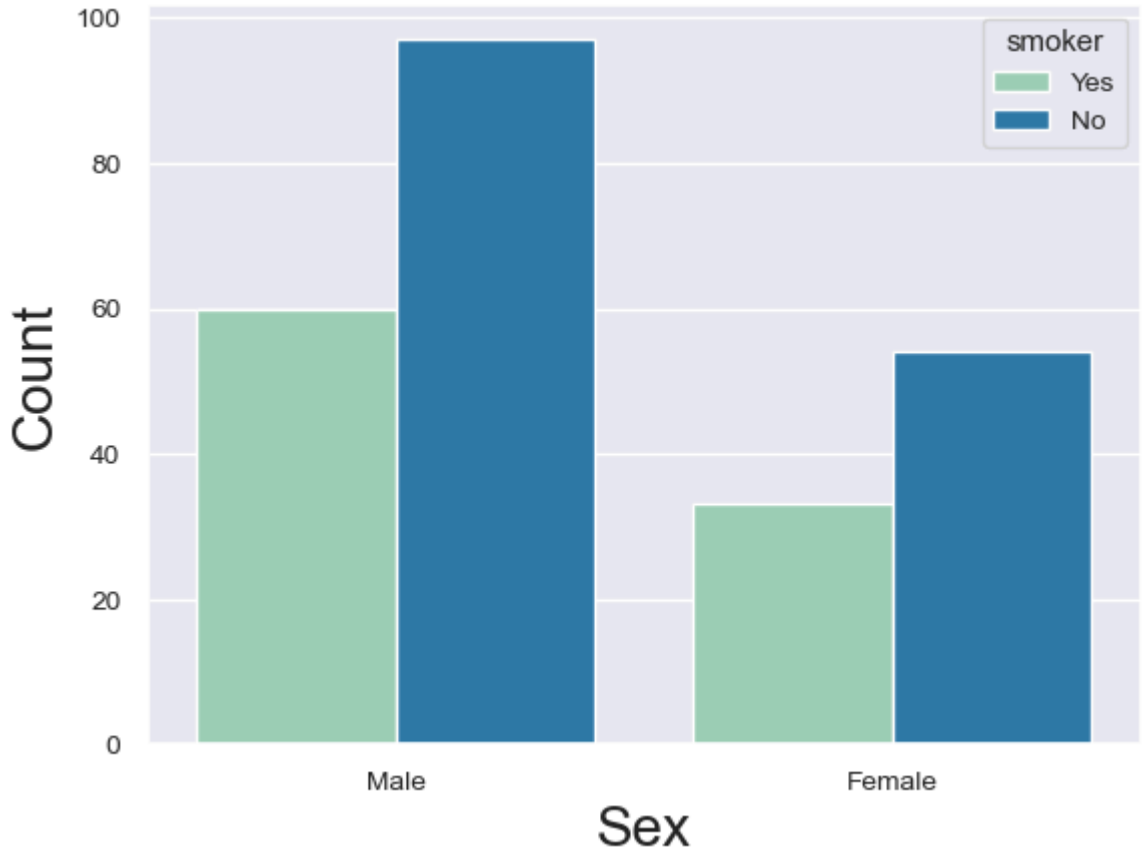
Hue = on behalf or respect

Palette = colour


ValueError: 'a' is not a valid value for name; supported values are 'Accent', 'Accent_r', 'Blues', 'B
lues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_
r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges
_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG',
'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Pur
ples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlG
n_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_
r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'Yl
OrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'bone_r',
'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r',
r', 'copper_r', 'crest', 'crest_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'flare',
'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_heat', 'gist_heat_r',
ist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_yarg',
r', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r',
r', 'icefire', 'icefire_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r',
_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r',
'prism', 'prism_r', 'rainbow', 'rainbow_r', 'rocket', 'rocket_r', 'seismic',
'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'tab20', 'tab20_r',
0c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight',
ted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'vlag', 'vlag_r', 'winter',
WsCUBE TECH


r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges
_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG',
'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Pur
ples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlG
n_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_
r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'Yl
OrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'bone_r', 'brg',
'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'coppe
r', 'copper_r', 'crest', 'crest_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'flare', 'flare_r',
'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_heat', 'gist_heat_r', 'gist_ncar', 'g
ist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_yarg',
r', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r', 'hsv',
r', 'icefire', 'icefire_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'mako
_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma',
'prism', 'prism_r', 'rainbow', 'rainbow_r', 'rocket', 'rocket_r', 'seismic', 'seismic_r',
'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'tab20', 'tab20_r', 'tab20b',
0c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight', 'twilight_r',
ted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'vlag', 'vlag_r', 'winter',
During handling of the above exception, another exception occurred:

```
sns.countplot(x="sex",data=var,hue="smoker",palette="YlGnBu"
,saturation=0.8)

plt.xlabel("Sex",fontsize=20)
plt.ylabel("Count",fontsize=20)
plt.show()
```



violin plot

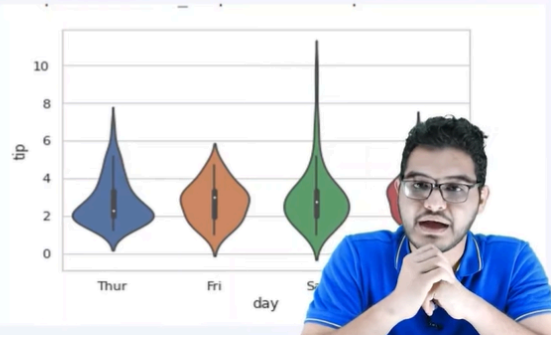
Violin Plot

Violin plot In Seaborn

WsCube Tech

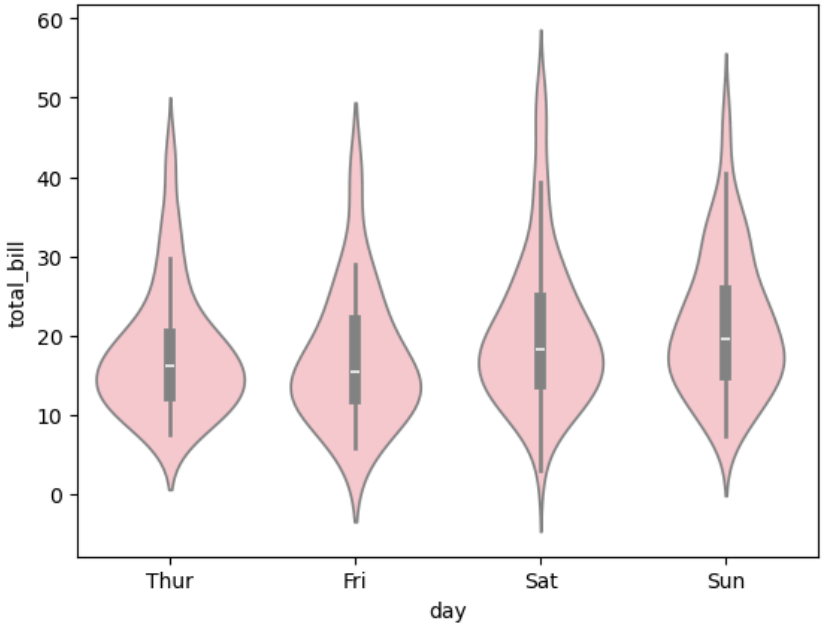
A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared.

```
import seaborn
seaborn.set(style = 'whitegrid')
f = seaborn.load_dataset("fmri")
seaborn.violinplot(x ="",y ="",data = f)
```



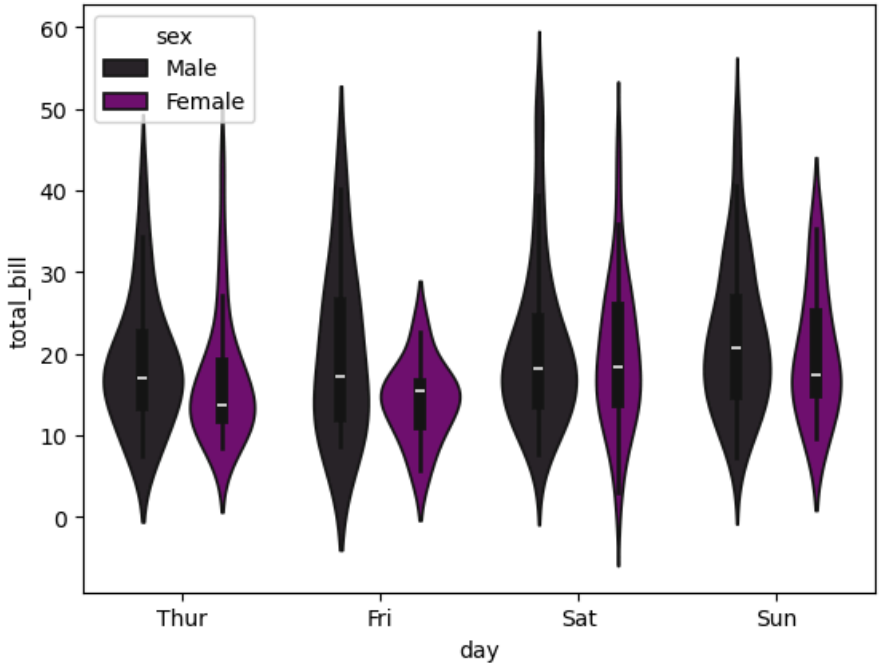
1) Normal

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

dt = sns.load_dataset("tips")
dt

sns.violinplot(x = "day",y="total_bill",data=dt,color="pink"
)
plt.show()
```

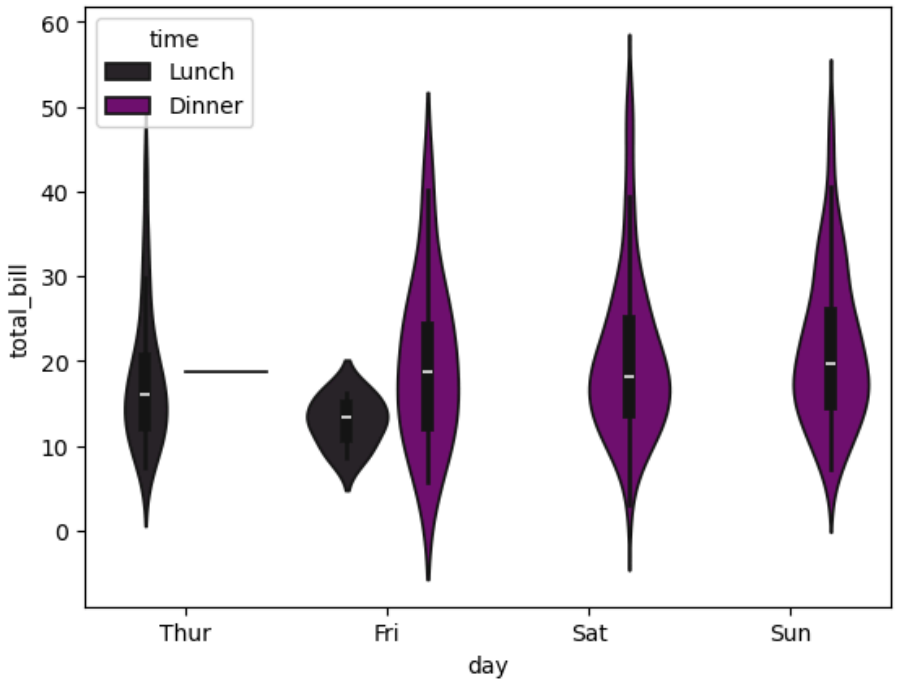


2) hue

```
sns.violinplot(x="day",y="total_bill",data=dt,color="purple"
,hue="sex")
plt.show()
```



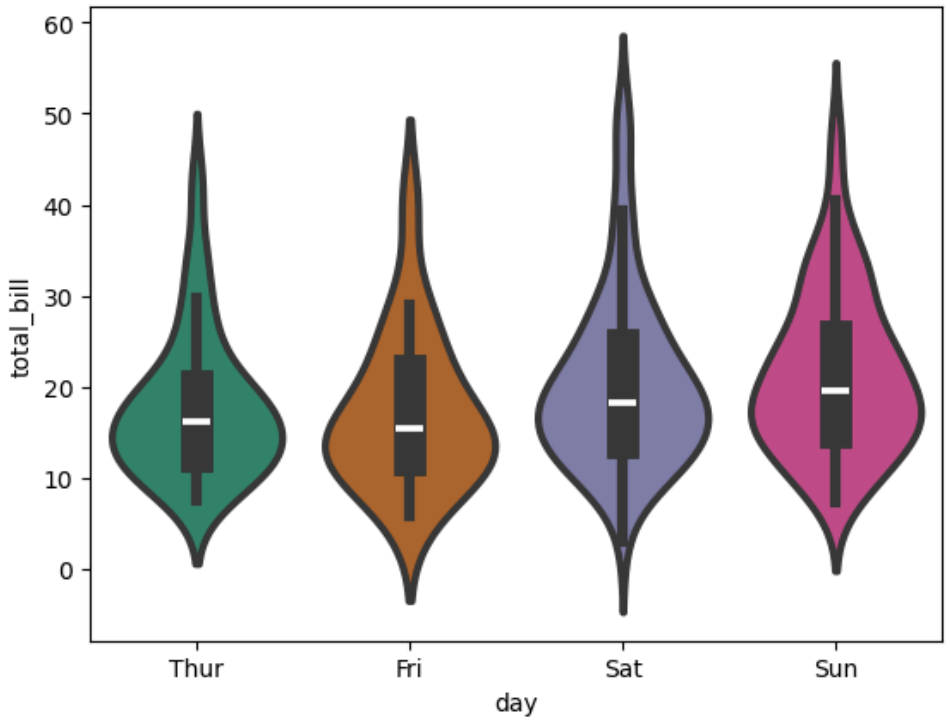

```
sns.violinplot(x="day",y="total_bill",data=dt,color="purple"
,hue="time")
plt.show()
```



3) Parameter

Linewidth - width of boundary of violine

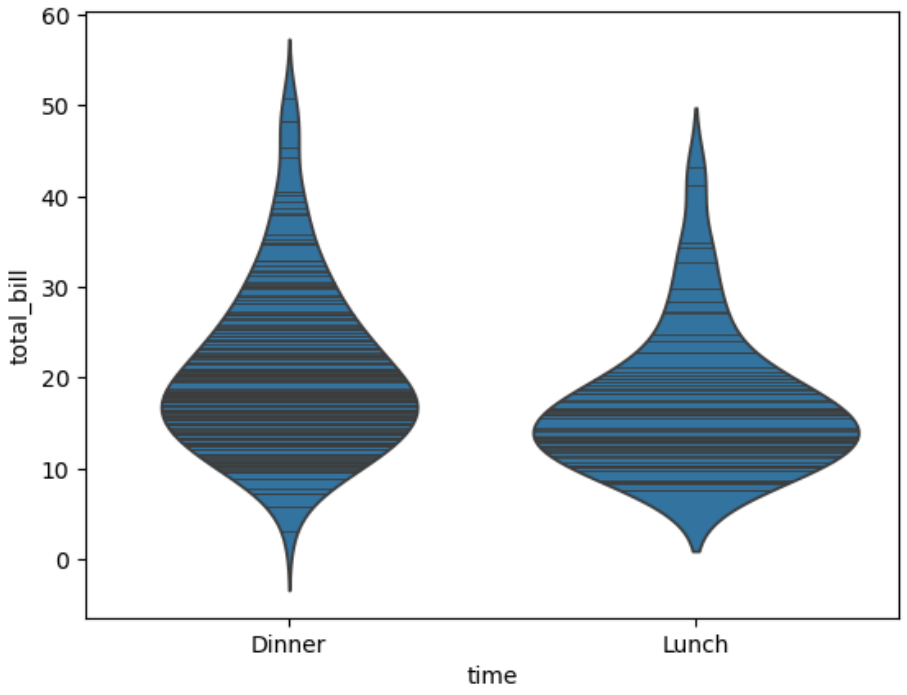
```
sns.violinplot(x="day",y="total_bill",data=dt,linewidth=3,palette="Dark2",saturation=0.6)
plt.show()
```



4) Parameter 2

Order and, **Inner** - style of inner .. `"quartile","point","box",None`...

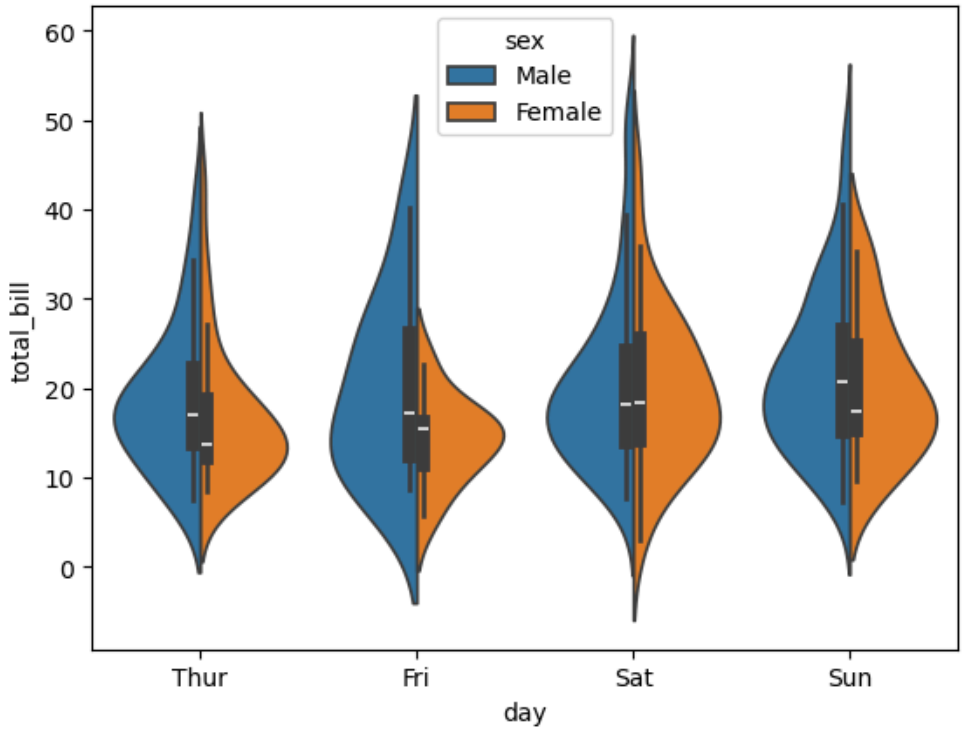
```
sns.violinplot(x="time",y="total_bill",data=dt,order=["Dinner","Lunch"],inner="stick")

# for inner - "quartile","point","box",None...

plt.show()
```



5) Split (two data visuals together)

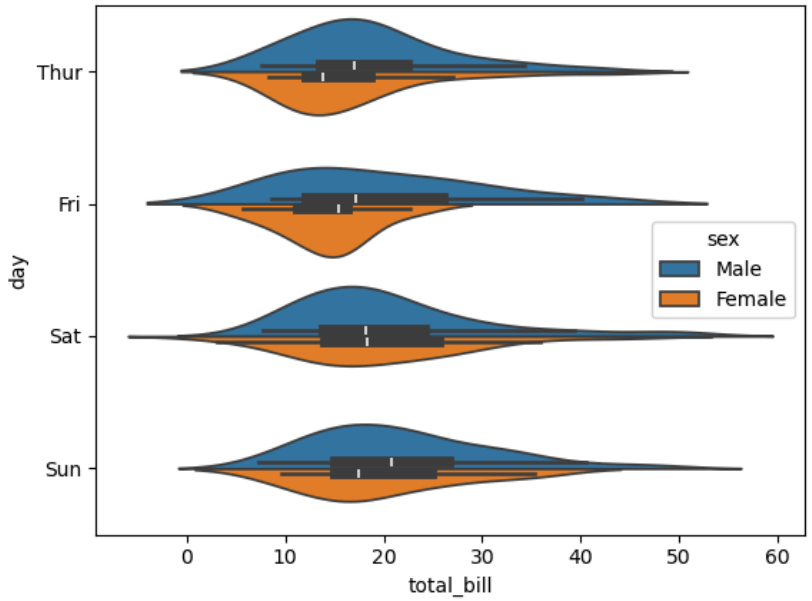
Split - split b/w two or more, **Scale -** variation of design like,
Scale - “width”, “area”, “count”

```
sns.violinplot(x="day",y="total_bill",data=dt,hue="sex",spli
t=True,scale="width")
plt.show()
```



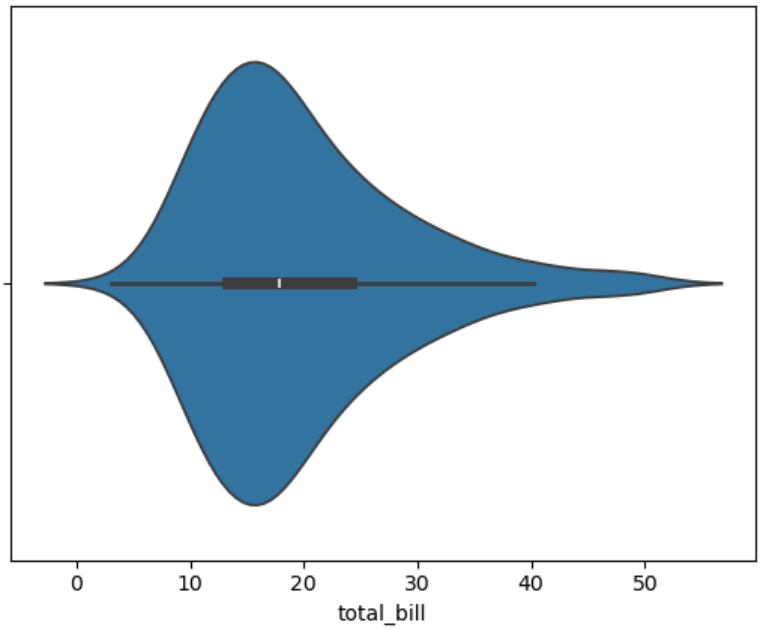
6) Orientation (Horizontal)

```
sns.violinplot(x="total_bill",y="day",data=dt,hue="sex",split=True)
plt.show()
```



7) Single Violine

```
sns.violinplot(x=dt["total_bill"])
plt.show()
```



if want change orientation then,(y=dt["total_bill"])

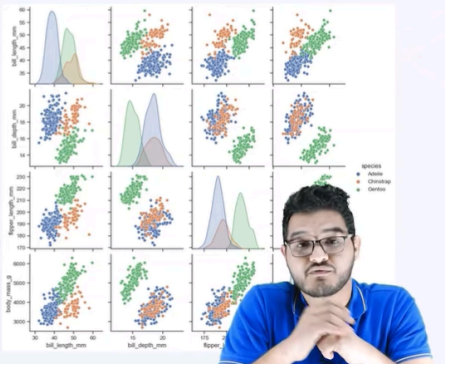
Pair Plot

Pair Plot

Pair Plot In Seaborn

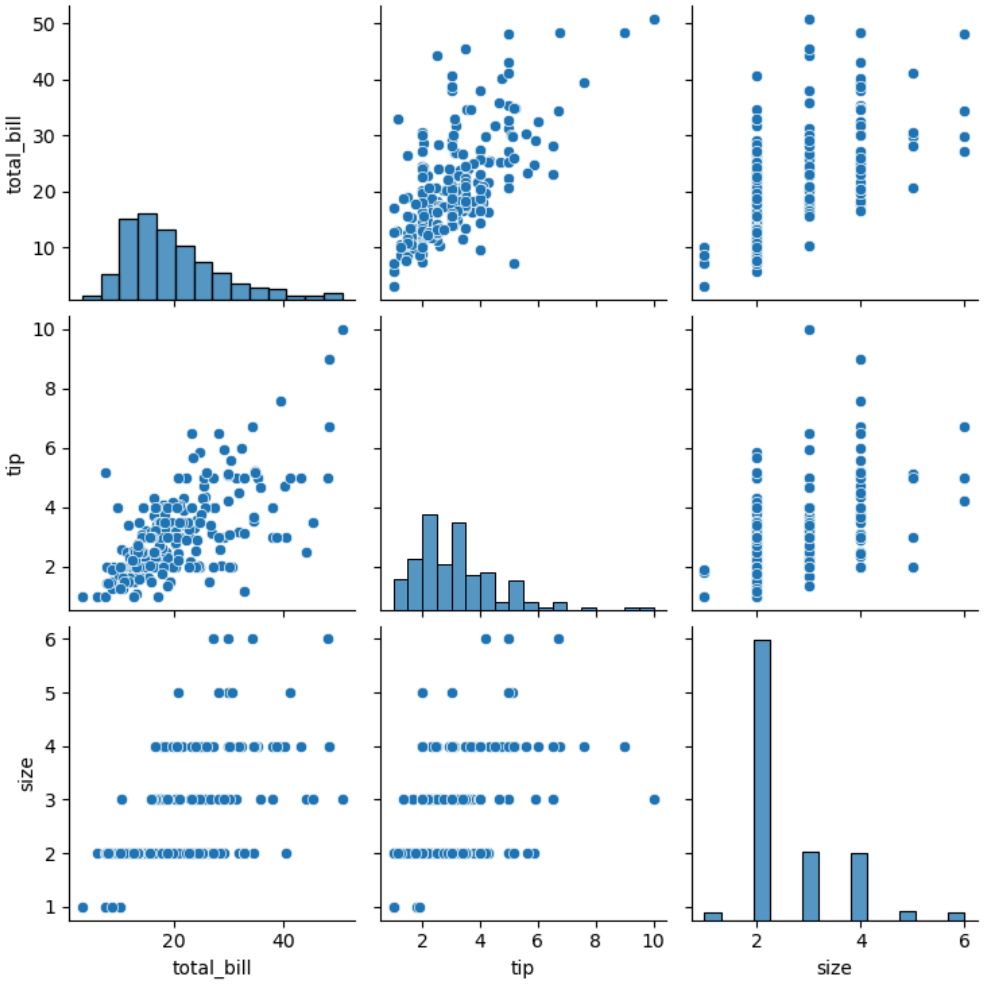
WsCube Tech

```
import seaborn
import matplotlib.pyplot as plt

df = seaborn.load_dataset(' ')
seaborn.pairplot(df)
plt.show()
```



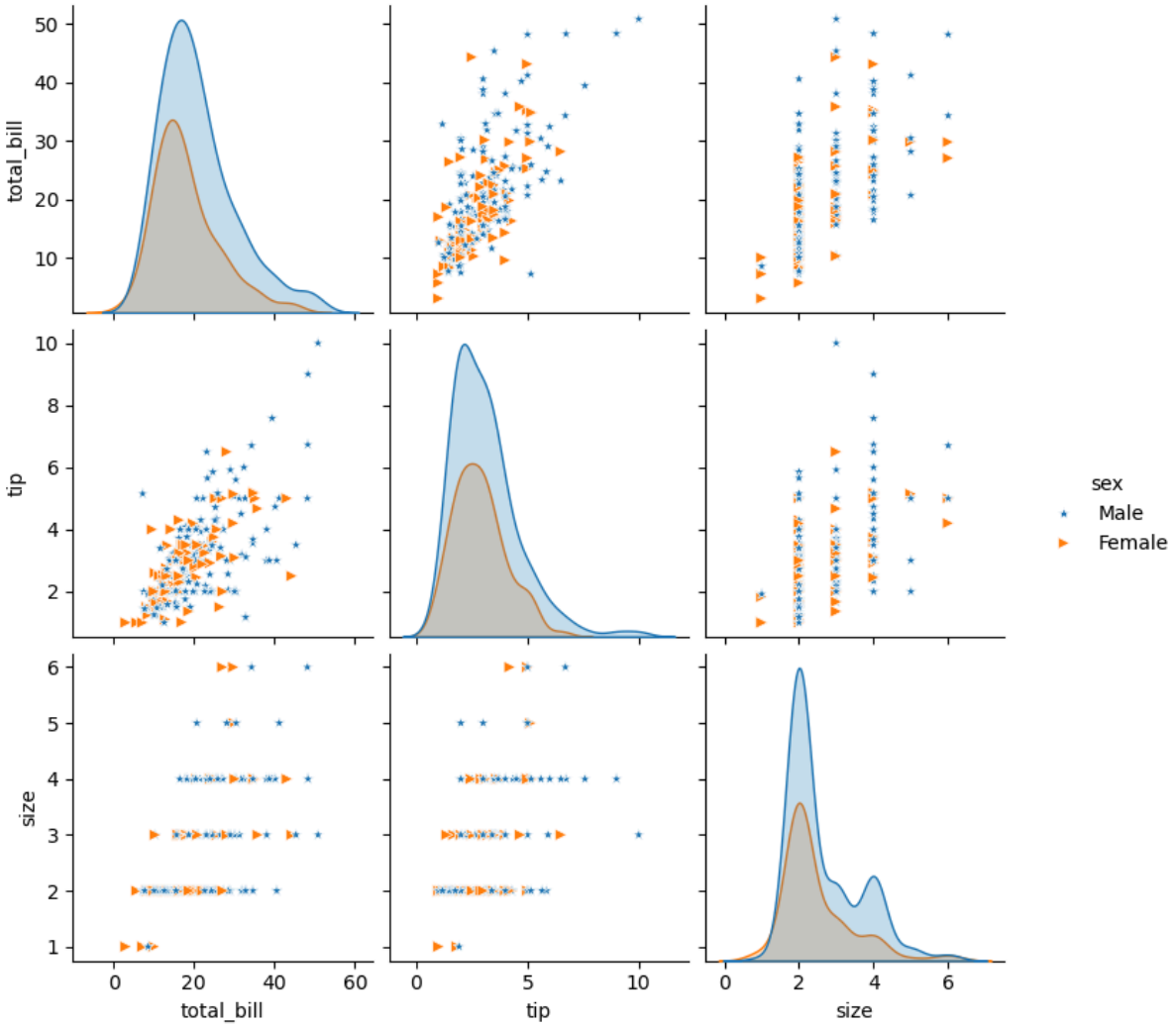
1) Normal

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dt = sns.load_dataset("tips")
Dt
```



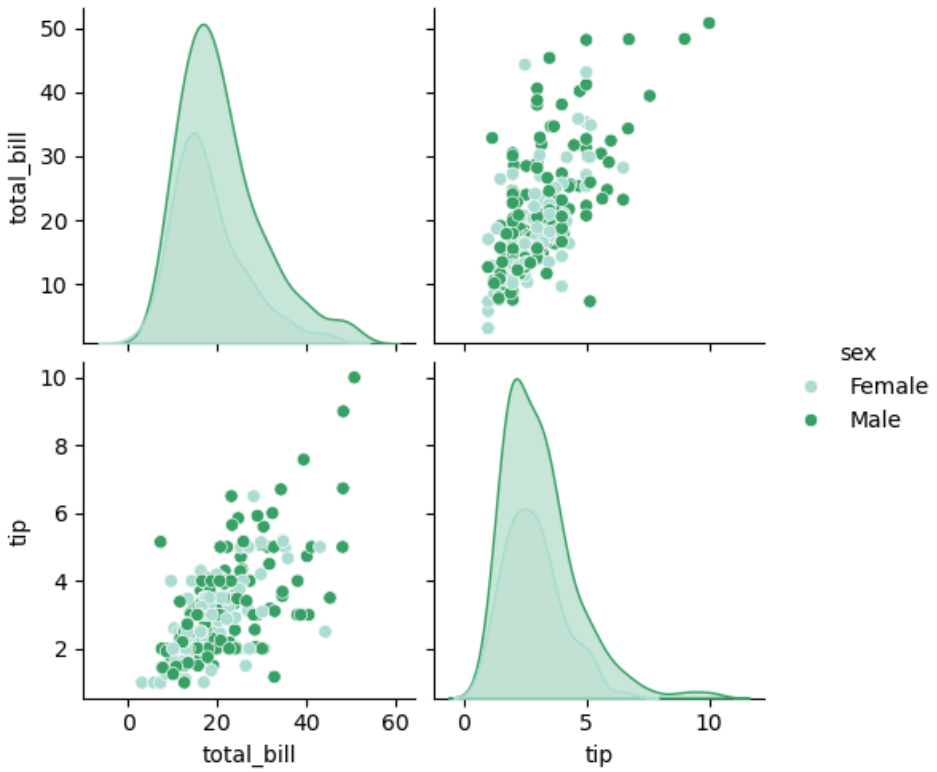
2) Hue, and marker

```
sns.pairplot(dt,hue="sex",markers=["*",">"])
plt.show()
```



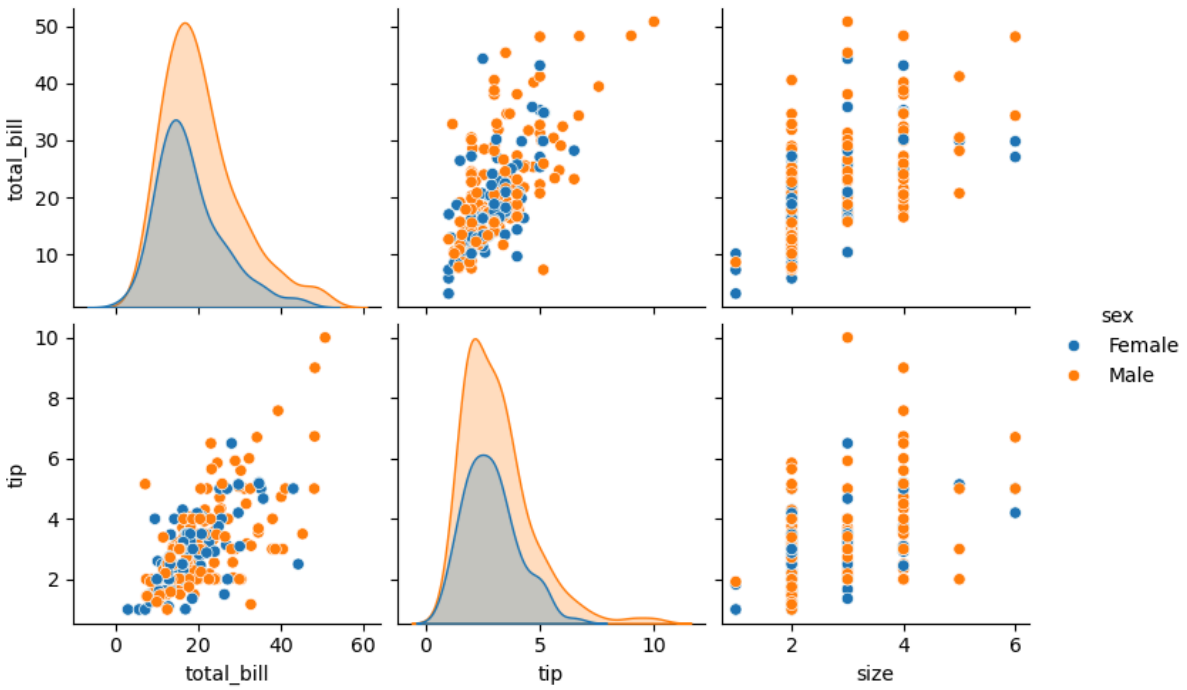
3) Fixed no of variables (Like vars =),

```
sns.pairplot(dt,vars=["total_bill","tip"],hue="sex",hue_orde
r=["Female","Male"],palette="BuGn")

plt.show()
```



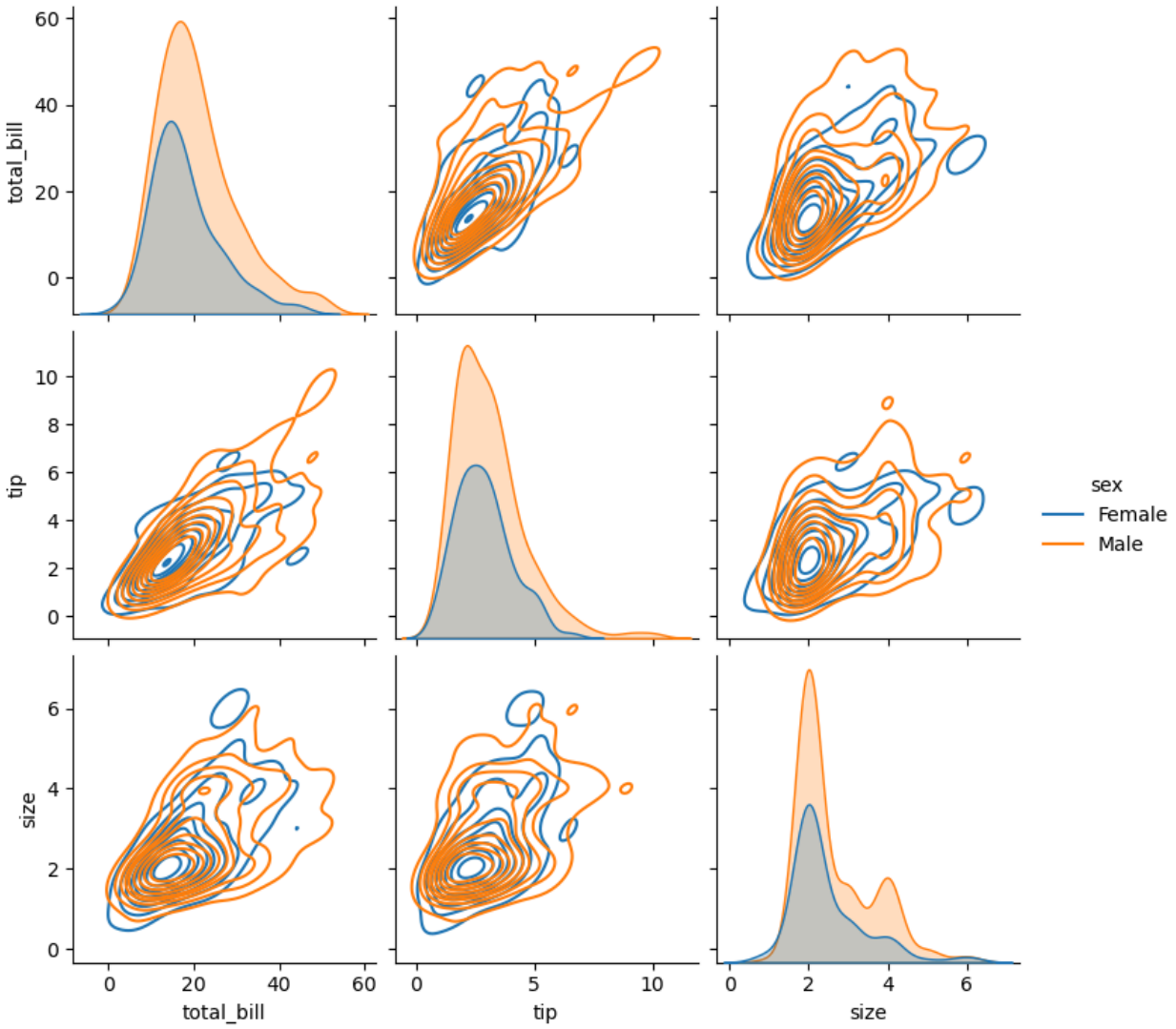
4) Fix x or y axis according you

```
sns.pairplot(dt,hue="sex",hue_order=["Female","Male"],y_vars
=["total_bill","tip"])

plt.show()
```



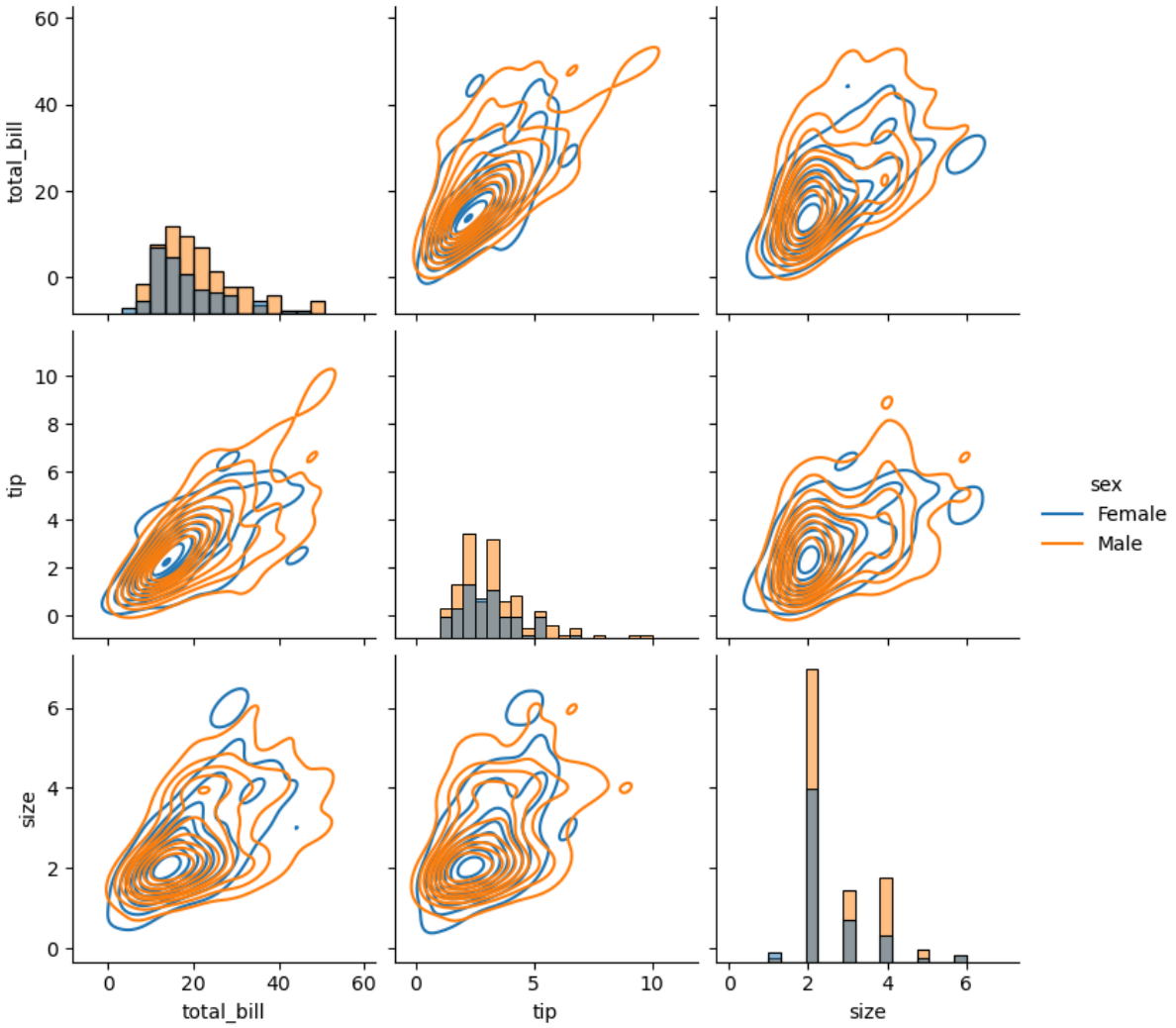
5) Kind - reg (regresion) , kde (Kernel Density Estimate) etc

```
sns.pairplot(dt,hue="sex",hue_order=["Female","Male"],kind="
kde")
#kind - 'scatter', 'kde', 'hist', 'reg'
plt.show()
```



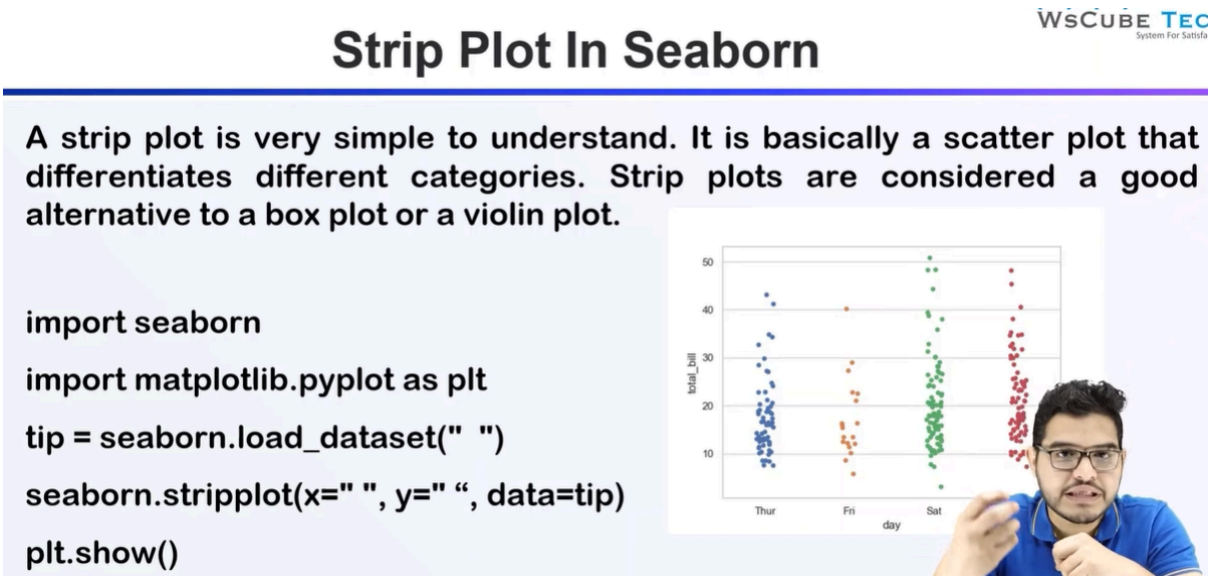
6) Dia_kind - fix diagonal with (kde or hist)

```
sns.pairplot(dt,hue="sex",hue_order=["Female","Male"],kind="
kde",diag_kind="hist")

plt.show()
```



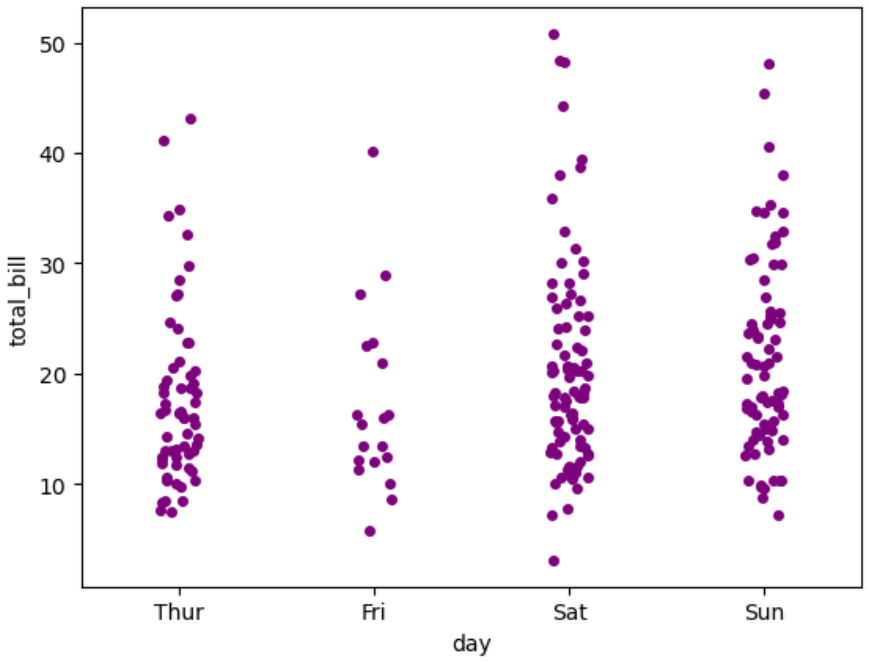
Strip Plot

Strip Plot


WsCube Tec
Strip Plot In Seaborn
A strip plot is very simple to understand. It is basically a scatter plot that differentiates different categories. Strip plots are considered a good alternative to a box plot or a violin plot.
import seaborn
import matplotlib.pyplot as plt
tip = seaborn.load_dataset(" ")
seaborn.stripplot(x=" ", y=" ", data=tip)
plt.show()
total_bill
day
Thur
Fri
Sat

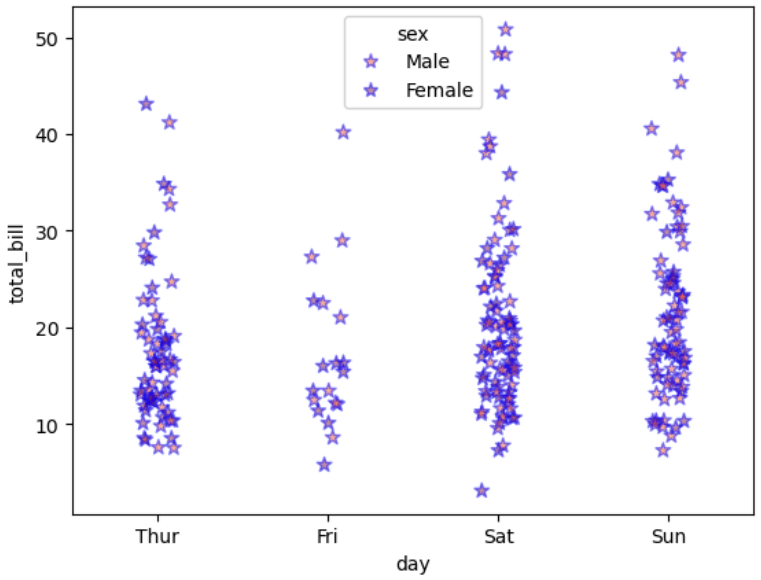
1) Normal

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns


dt = sns.load_dataset("tips")
Dt

sns.stripplot(x="day",y="total_bill",data=dt,color="purple")
plt.show()
```



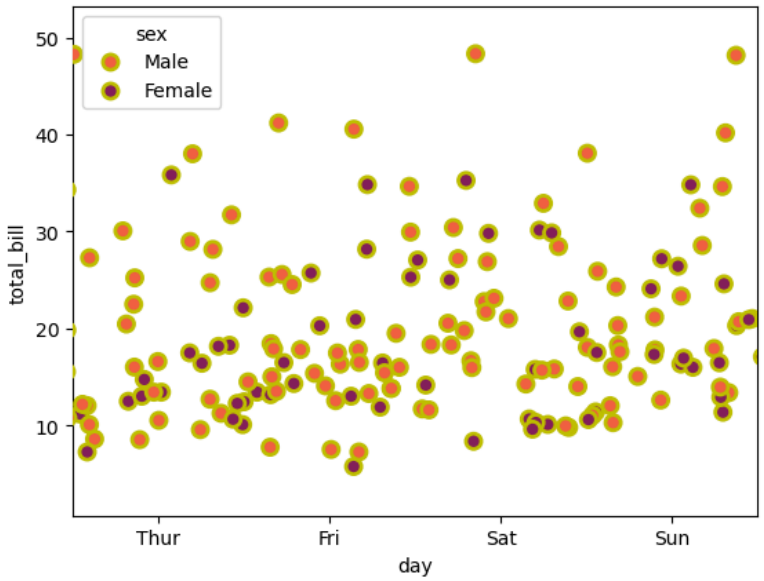
2) With perimeter

```
sns.stripplot(x="day",y="total_bill",data=dt,hue="sex",
              palette='rocket_r',linewidth=1,edgecolor="b",
              size=8,alpha=0.5,marker="*")

plt.show()
```



- ## Jitter

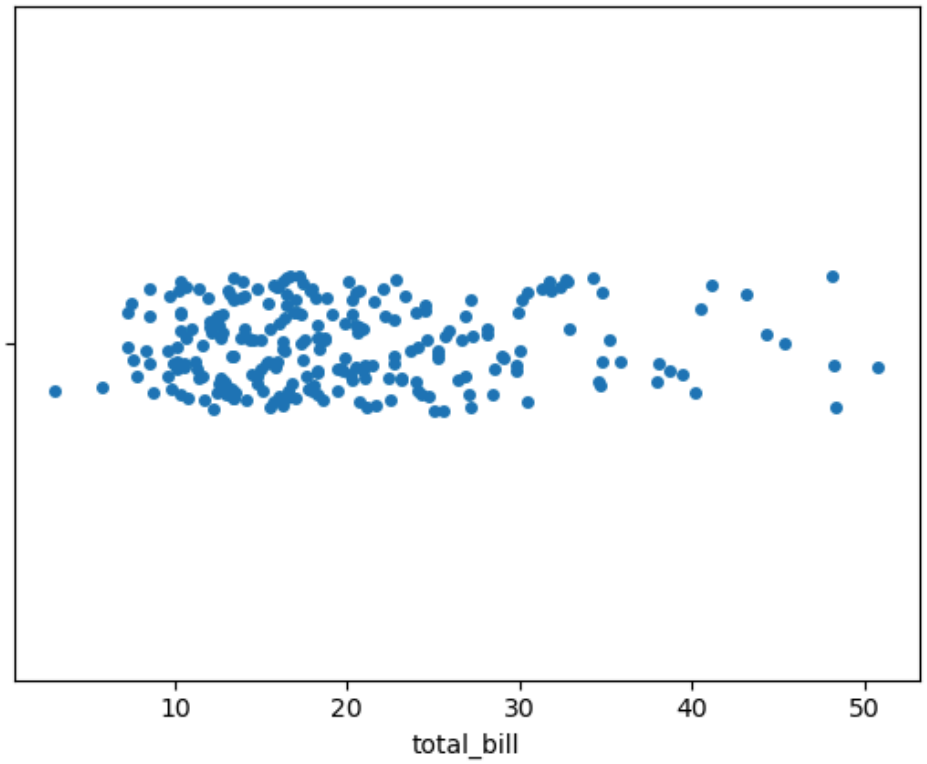
```
y = {"Male":"*","Female":">"}
sns.stripplot(x="day",y="total_bill",data=dt,hue="sex",
              palette='rocket_r',linewidth=2,edgecolor="y",
              jitter=3,size=8)
plt.show()
```



3) One strip plot

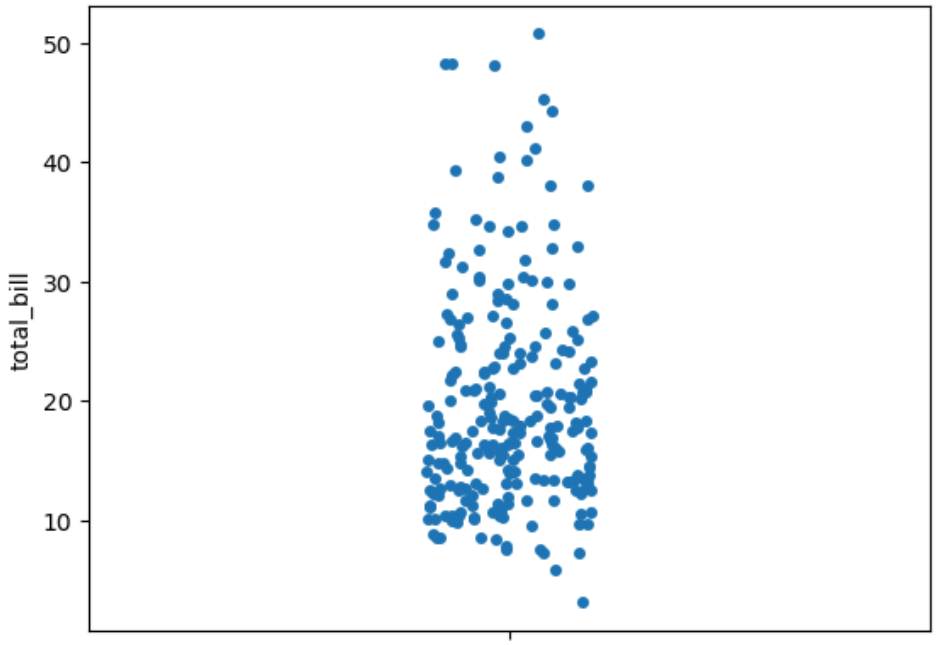
- **Horizontally**

```
sns.stripplot(x="total_bill",data=dt)
plt.show()
```



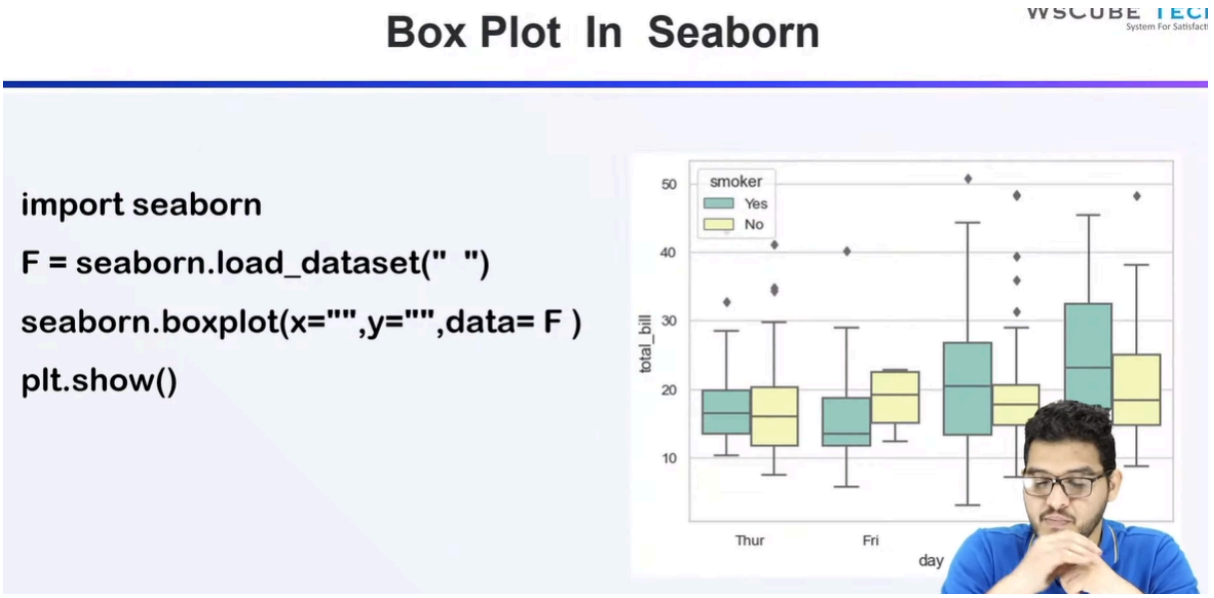
- **Vertically**

```
sns.stripplot(y=dt["total_bill"])
plt.show()
```



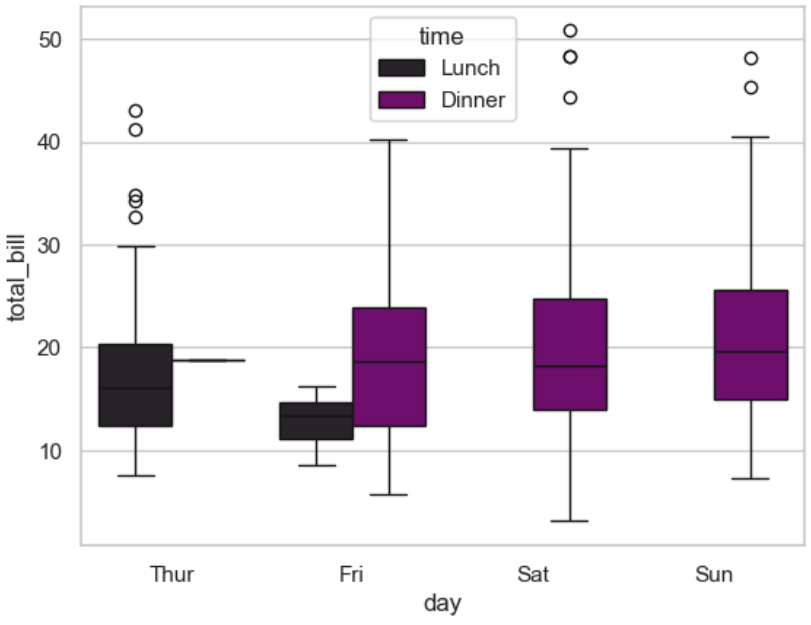
Box plot

Box Plot



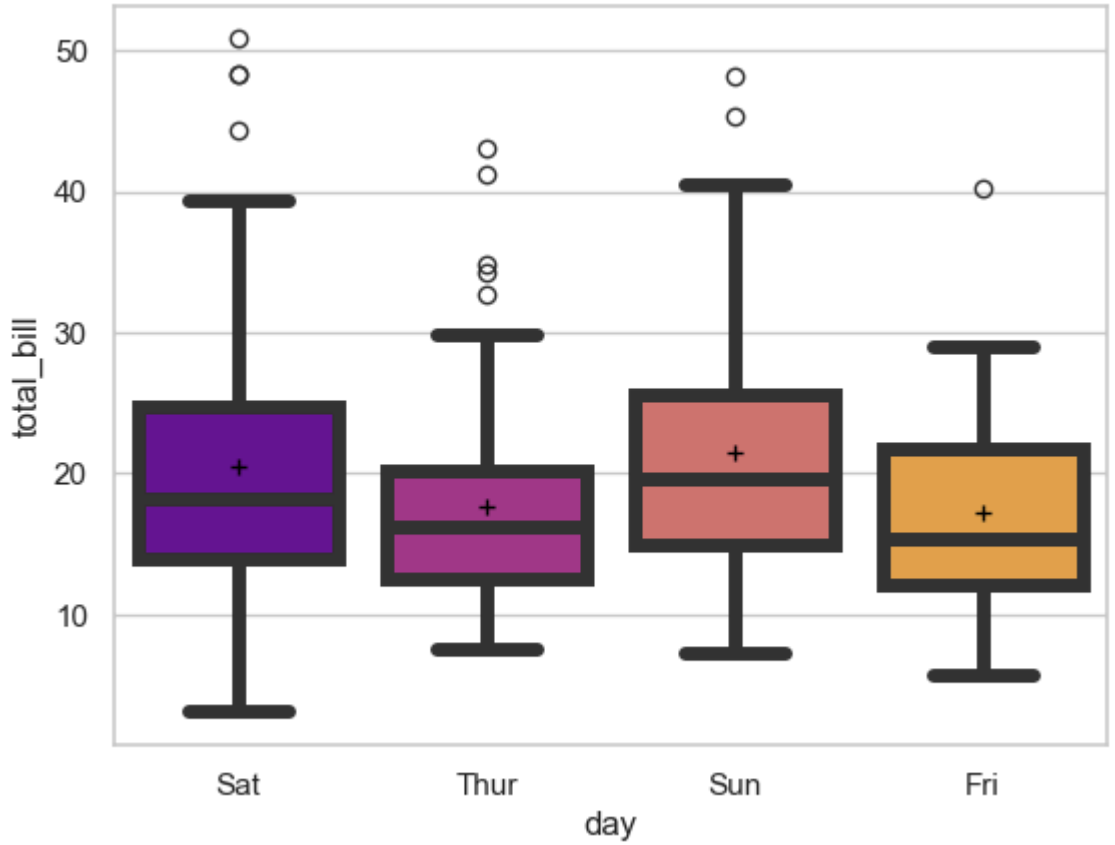
1) Normal

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dt = sns.load_dataset("tips")
dt

sns.set(style="whitegrid")
sns.boxplot(x="day",y="total_bill",data=dt,color="purple",hue="time")
```



2) With perimeter

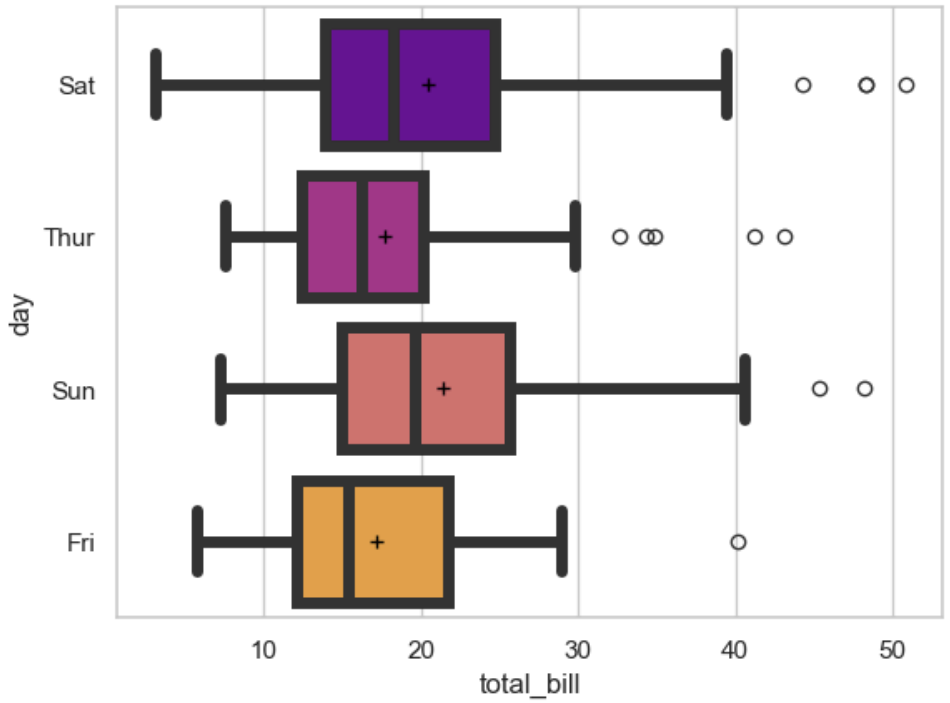
```
sns.set(style="whitegrid")

sns.boxplot(x="day",y="total_bill",data=dt,order=["Sat","Thur","Sun","Fri"],showmeans=True,meanprops={"marker":"+","markeredgecolor":"black"},linewidth=5,palette="plasma",orient="v")

plt.show()
```



For horizontal need both axis should be integer… but inspite of it there is some ways to do it -

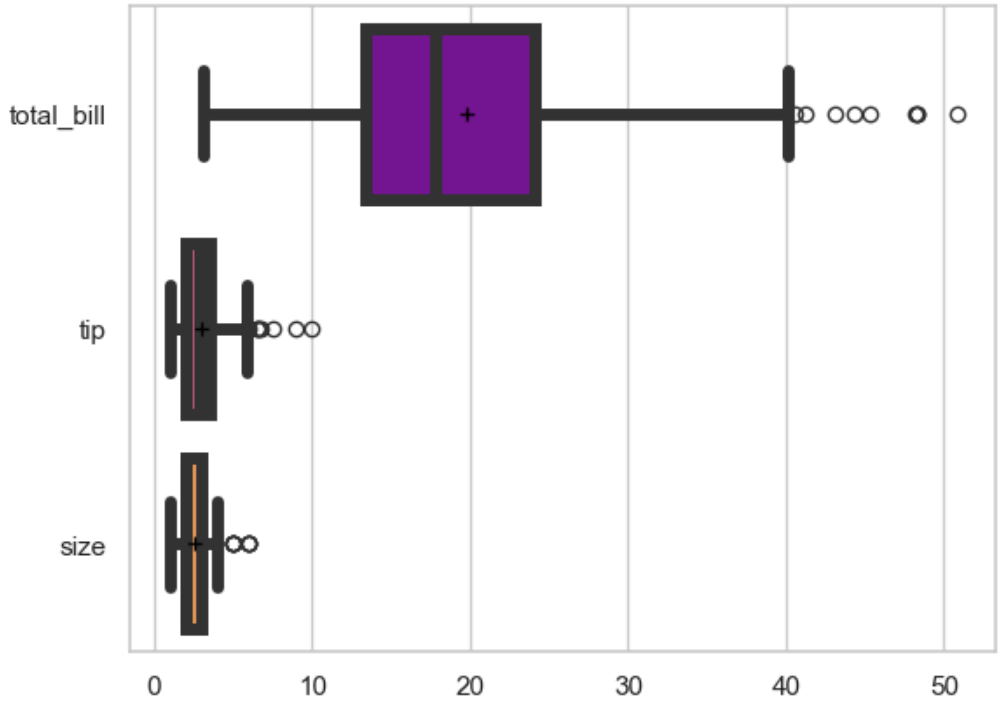
- ## For horizontal

```
sns.set(style="whitegrid")
sns.boxplot(x="total_bill",y="day",data=dt,showmeans=True,meanprops={"marker":"+","markeredgecolor":"black"},linewidth=5,palette="plasma")

plt.show()
```



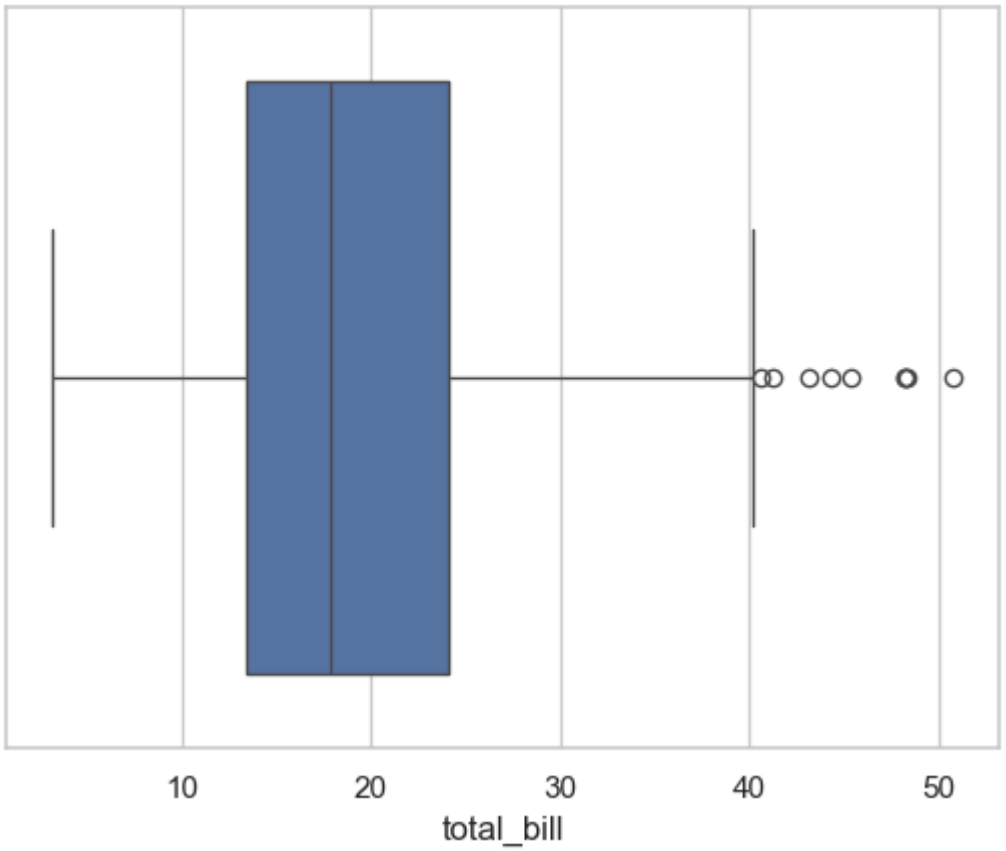
- **For horizontal another method**

```
sns.set(style="whitegrid")

sns.boxplot(data=dt,
showmeans=True,meanprops={"marker":"+","markeredgecolor":"bl
ack"},linewidth=5,palette="plasma",orient="h")

plt.show()
```



3) Single box plot

```
sns.set(style="whitegrid")
sns.boxplot(x= dt["total_bill"])

plt.show()
```



Factor plot

Cat Plot

Factor Plot In Seaborn

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv(' ')
sns.factorplot(x ='', y ='', data = df)
plt.show()
```

Seaborn have a `factorplot` function, but it has been deprecated and replaced by the more flexible `sns.catplot()` function.

Cat Plot In Seaborn

```
import seaborn as sns
exercise = sns.load_dataset("")
g = sns.catplot( x="", y="", data=exercise, kind = "" )
```

Cat Plot In Seaborn

Categorical scatterplots with catplot

- stripplot() – with kind="strip"
- swarmplot() – with kind="swarm"

Categorical estimate plots with catplot

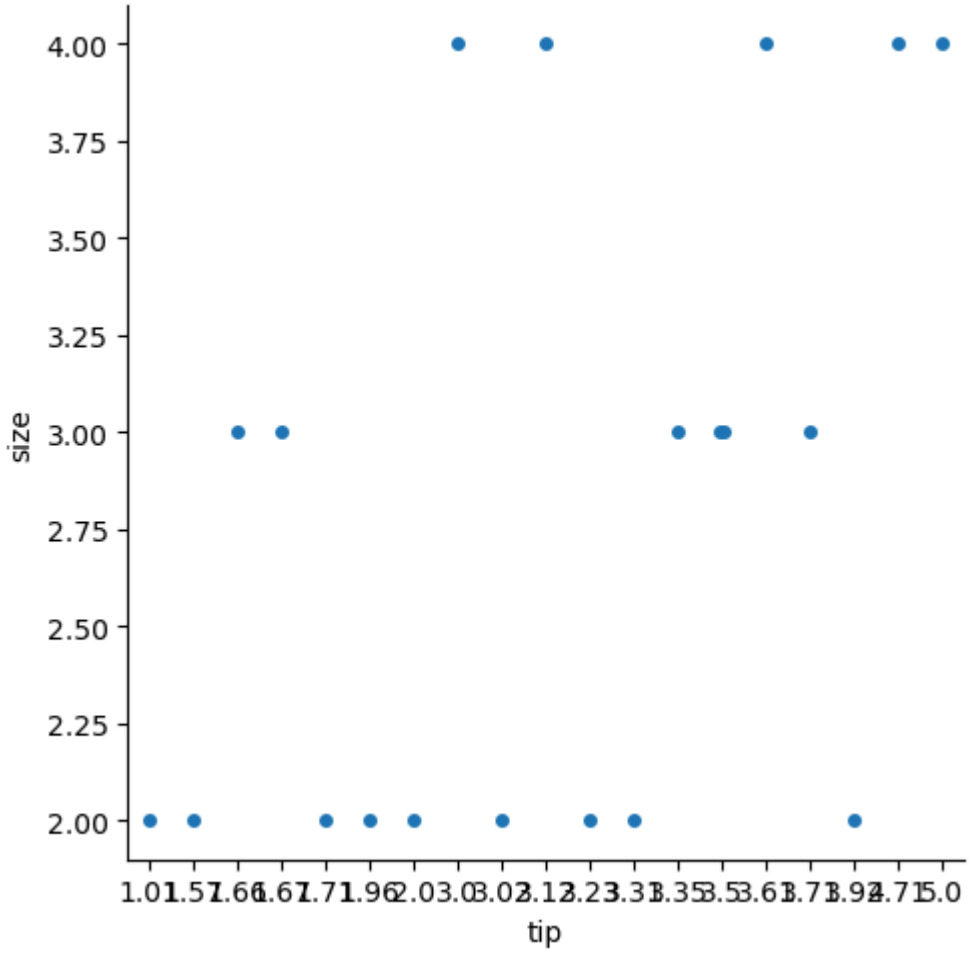
- pointplot() – with kind="point"
- barplot() – with kind="bar"
- countplot() – with kind="count"

Categorical distribution plots with catplot

- boxplot() – with kind="box"
- violinplot() – with kind="violin"
- boxenplot() – with kind="boxen"

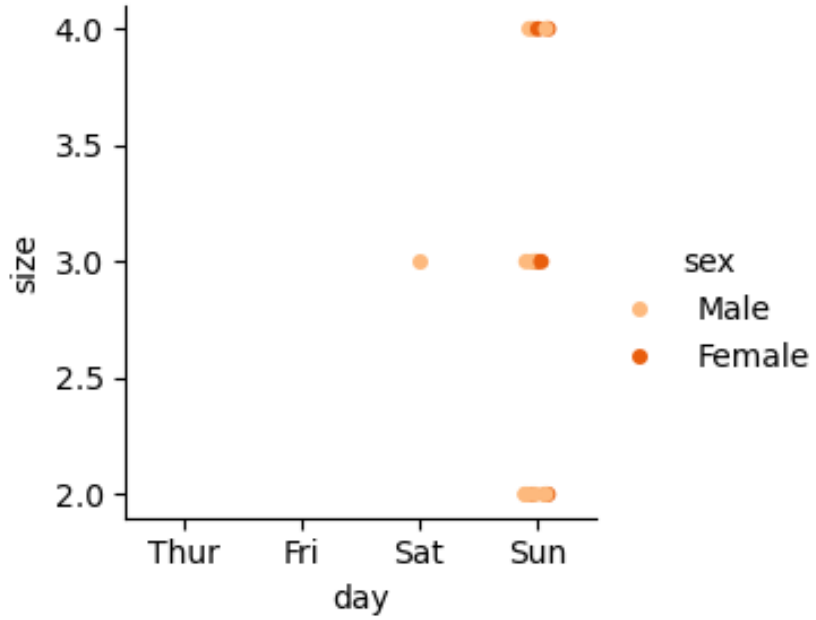
1) Normal

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dt = sns.load_dataset("tips")
x = dt.head(20)
x

sns.catplot(x="tip",y="size",data=x)
Plt.show
```



2) With perimeter

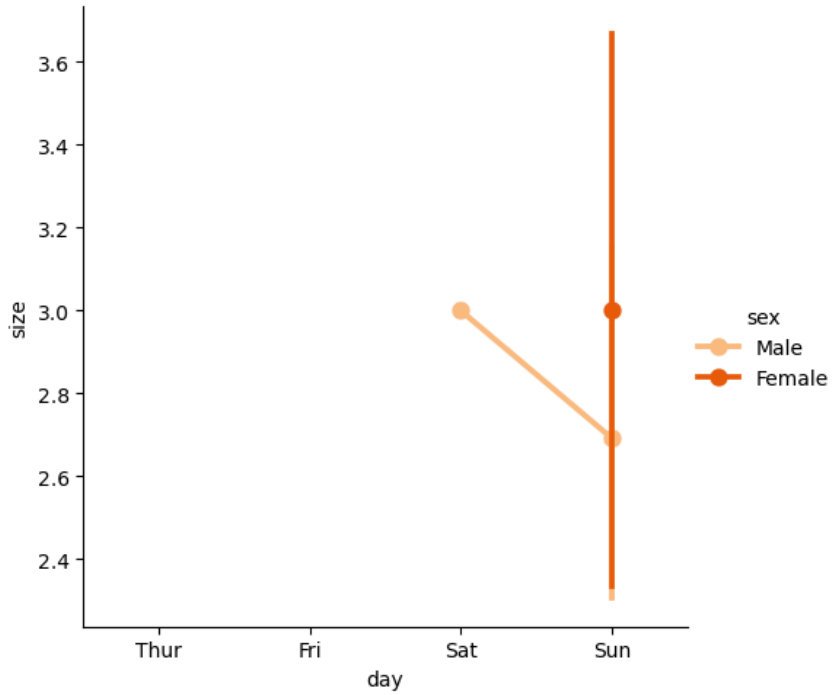
```
sns.catplot(x="day",y="size",data=x,hue="sex",palette="Orang
es",height=3)
Plt.show
```



3) By kind

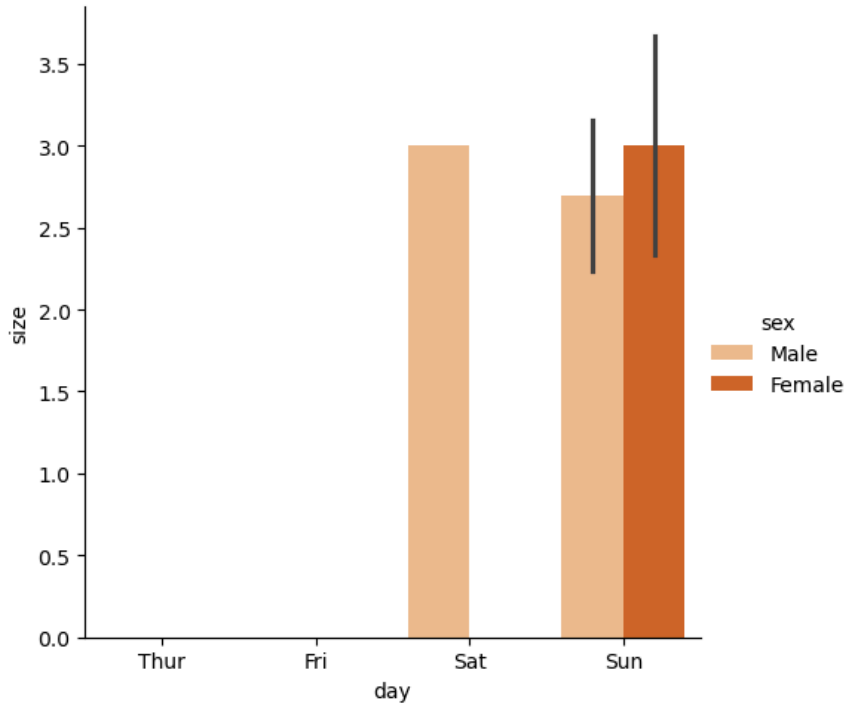
- **“Point”**

```
sns.catplot(x="day",y="size",data=x,hue="sex",palette="Oranges",kind="point")
plt.show
```



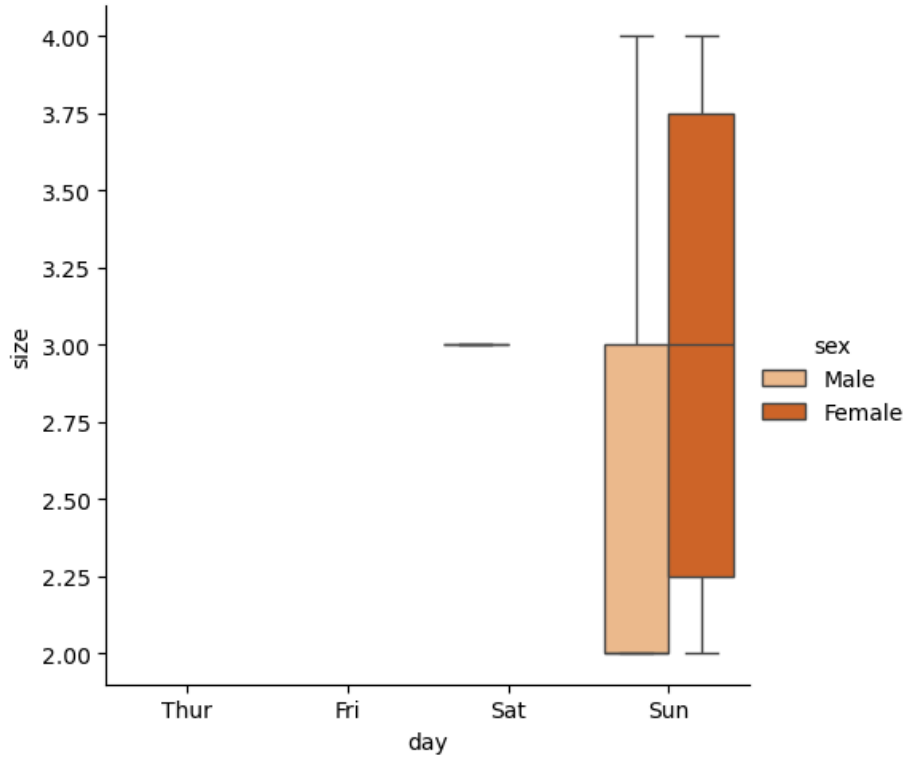
- **“Bar”**

```
sns.catplot(x="day",y="size",data=x,hue="sex",palette="Oranges",kind="bar")
Plt.show
```



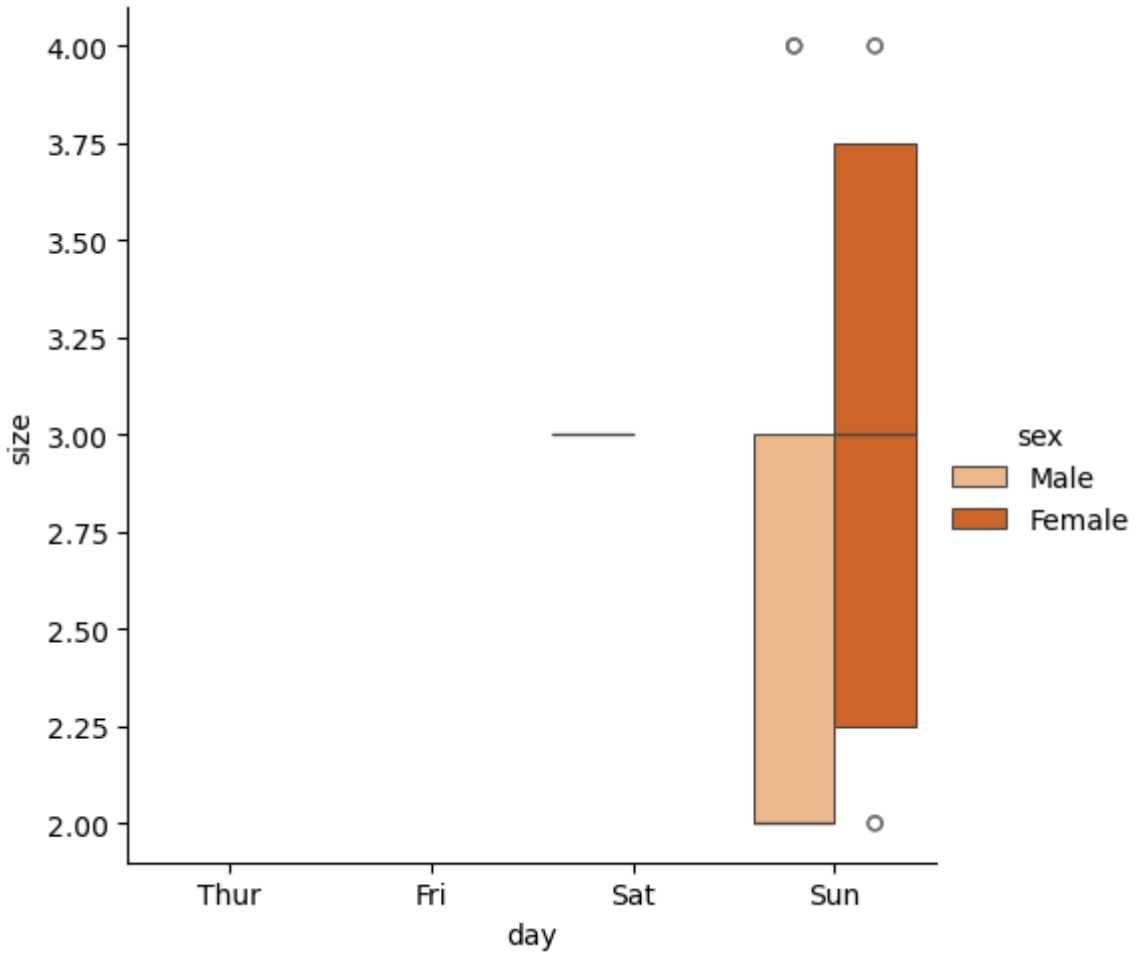
- **“Box”**

```
sns.catplot(x="day",y="size",data=x,hue="sex",palette="Oranges",kind="box")
plt.show
```



- **“Boxen”**

```
sns.catplot(x="day",y="size",data=x,hue="sex",palette="Oranges",kind="boxen")
plt.show
```



Styling plot

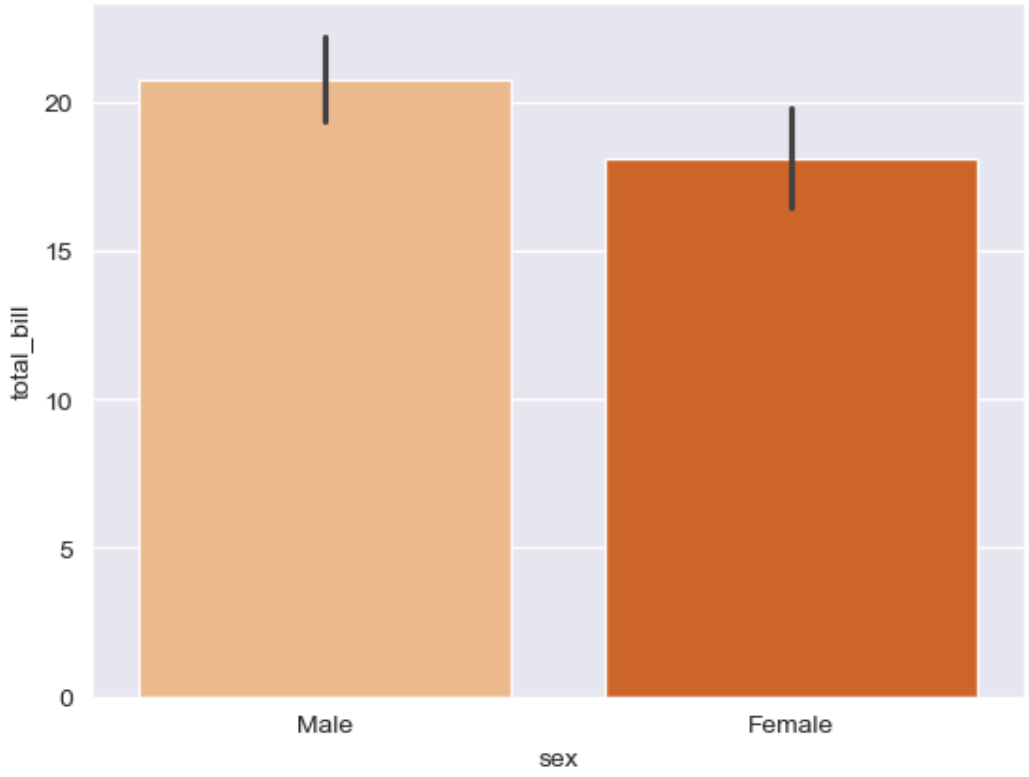
Styling Plot

Styling Plots In Seaborn

- Seaborn Figure Styles
- Removing Axes Spines
- Scale and Context

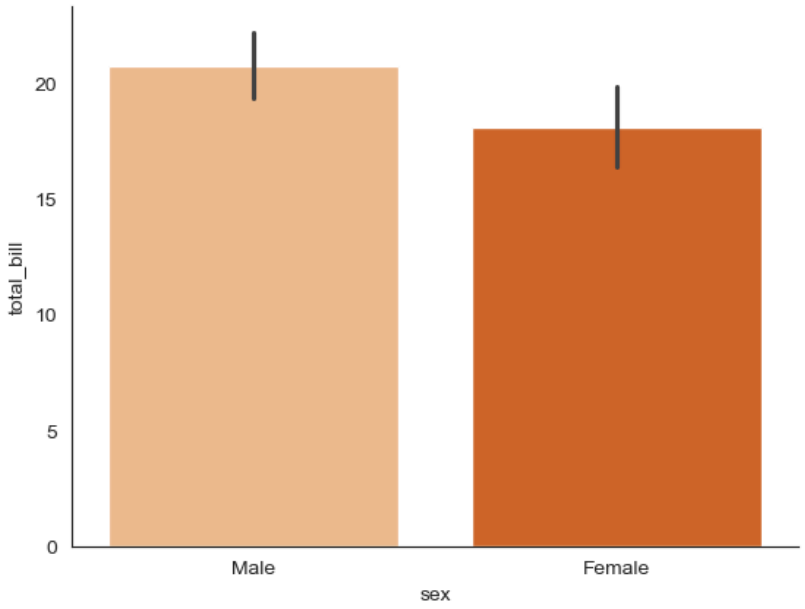
1) Seaborn Figure Style

```
sns.set_style("darkgrid") #- Dark, Darkgrid, White,
Whitegrid, ticks
sns.barplot(x="sex",y="total_bill",data=dt,palette="Oranges"
)
plt.show()
```



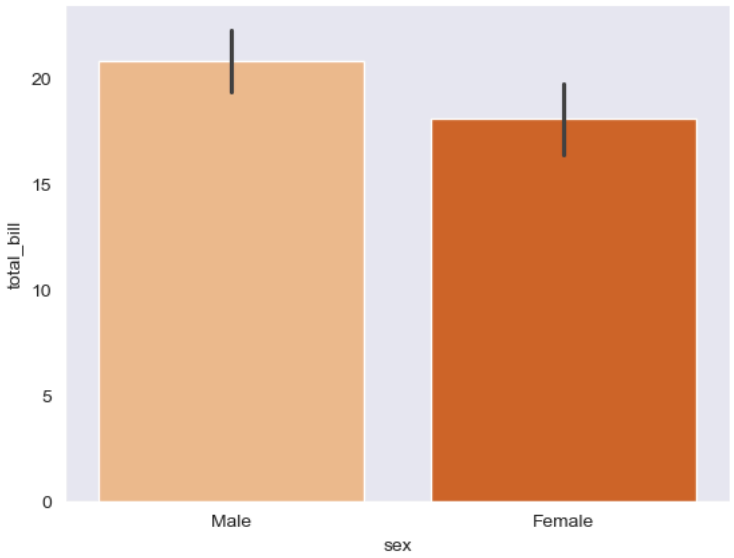
2) Removing Axes Spines

```
sns.set_style("white")

sns.barplot(x="sex",y="total_bill",data=dt,palette="Oranges"
)

sns.despine()
plt.show()
```



- **For dark**

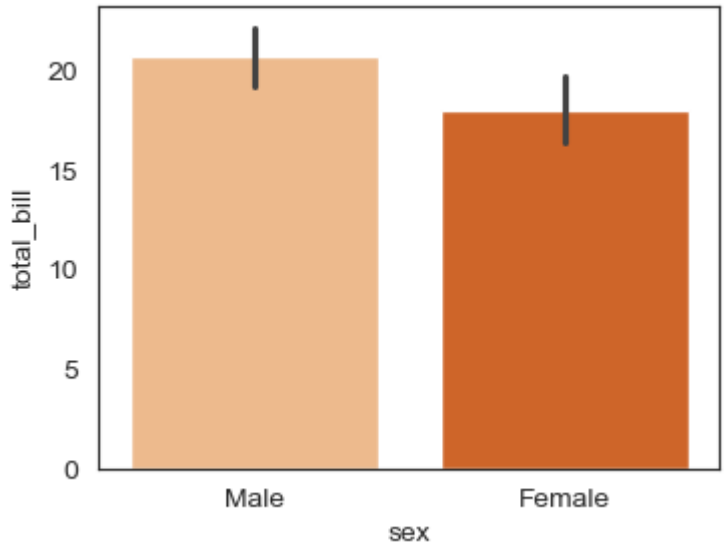
```
sns.set_style("dark")
sns.barplot(x="sex",y="total_bill",data=dt,palette="Oranges"
)
sns.despine()
plt.show()
```



3) Scale and Context

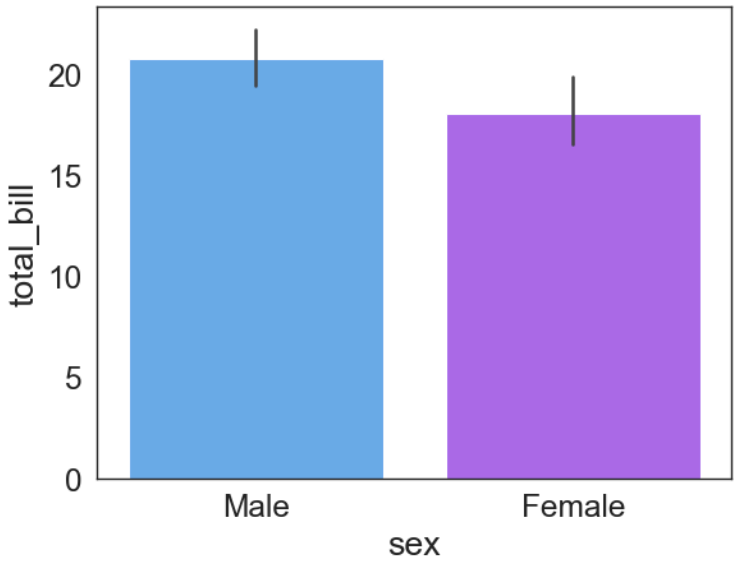
- ### Figsize

```
sns.set_style("white")
plt.figure(figsize=(4,3))
sns.barplot(x="sex",y="total_bill",data=dt,palette="Oranges"
)
plt.show()
```



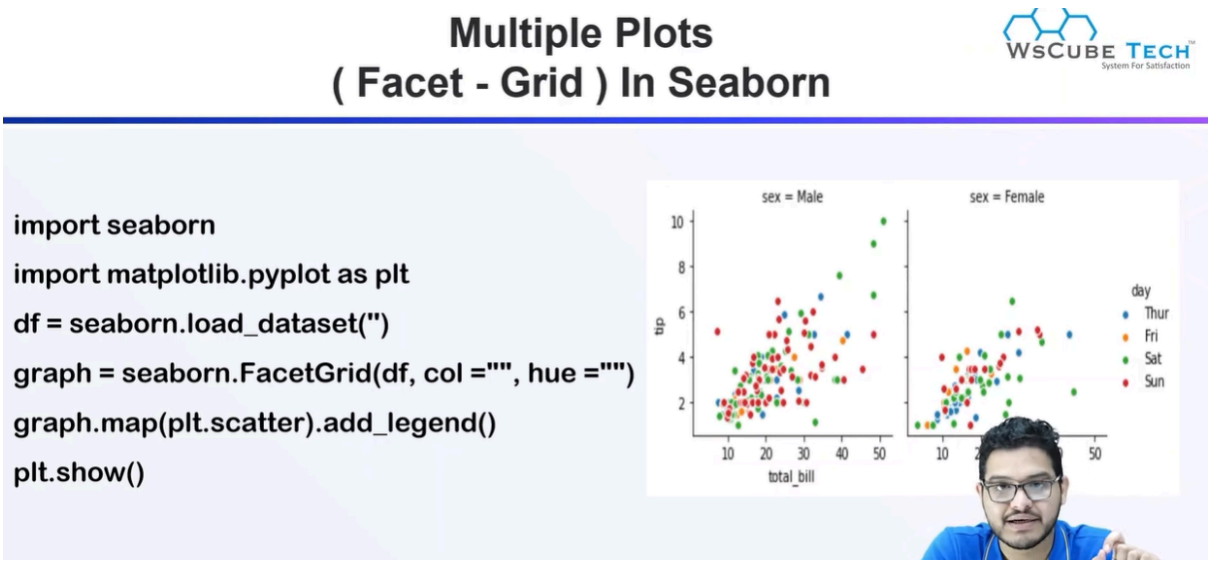
- ### sns.set_context

```
sns.set_style("white")
sns.set_context("paper",font_scale=2)
#notebook,talk,poster,paper
sns.barplot(x="sex",y="total_bill",data=dt,palette="cool")
plt.show()
```

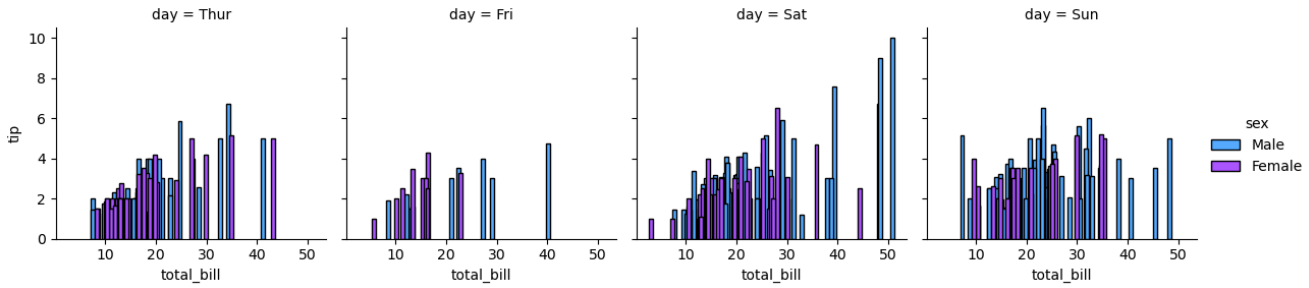


(Facet) Multiple Plot

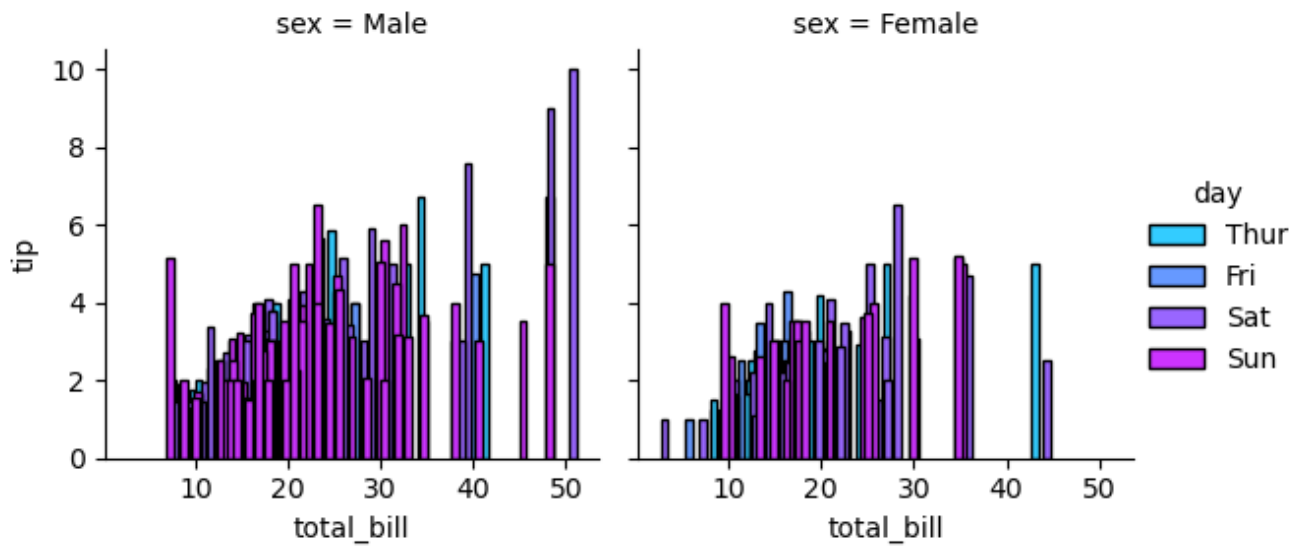
Multiple Plot (Facet Grid)




```
fg = sns.FacetGrid(dt,col="day",hue="sex",palette="cool")

fg.map(plt.bar,"total_bill","tip",edgecolor=
"black").add_legend()

plt.show()
```




```
fg = sns.FacetGrid(dt,col="sex",hue="day",palette="cool")

fg.map(plt.bar,"total_bill","tip",edgecolor=
"black").add_legend()

plt.show()
```



Explainations

1️⃣ Why do we create `fg = sns.FacetGrid(...)`?

fg = sns.FacetGrid(dt, col="sex", hue="day", palette="cool")

Here, `fg` is an **object** of type `FacetGrid`.

What is `FacetGrid`?

`FacetGrid` is used when you want to:

- Split your dataset into **multiple subplots**
- Based on **categorical variables**

In your case:

- `col="sex"` → creates **separate plots (columns)** for each value of `sex`
  - One plot for `Male`
  - One plot for `Female`
- `hue="day"` → further divides data **within each plot** by `day`
- `palette="cool"` → assigns colors for different `day` values

So `fg` represents the **entire grid of plots**, not a single chart.

> Think of `fg` as a *canvas with multiple empty subplots*, ready to be filled.

2️⃣ Why do we use `map()`?

fg.map(plt.bar, "total_bill", "tip", edgecolor="black")

Purpose of `map()`

`map()` tells Seaborn:

> "Apply this plotting function to **each subset of data** in the grid."

In other words:

- Seaborn **automatically splits** the data:
  - by `sex` → different columns
  - by `day` → different colors
- `map()` loops internally over each subset
- `plt.bar()` is applied **separately** to each facet

Why not just use `plt.bar()` directly?

Because `plt.bar()`:

- Does **not know** about:
  - faceting (`sex`)
  - color grouping (`day`)
- It would produce **only one plot**, not multiple subplots

`FacetGrid.map()`:

- Passes the **correct subset of data** to `plt.bar()`
- Ensures each subplot gets the right data

3️⃣ Why do we pass `plt.bar` inside `map()`?

fg.map(plt.bar, "total_bill", "tip")

`map()` expects:

map(plot_function, x_variable, y_variable)

Here:

- `plt.bar` → plotting function
- `"total_bill"` → x-axis
- `"tip"` → y-axis

Seaborn internally does something like:

```
for each sex:
    for each day:
        plt.bar(total_bill_subset, tip_subset)
```

You don't see the loop, but `map()` handles it for you.

4️⃣ Why do we need `fg` specifically?

Because:

- `FacetGrid` **stores**:
  - the dataset
  - the faceting logic (`col="sex"`)
  - the color grouping (`hue="day"`)
- `fg.map()` **depends on that stored configuration**

Without `fg`, Seaborn has no way to know:

- how many subplots to create
- how to split the data

5️⃣ Why do we call `.add_legend()`?

.add_legend()

Since `hue="day"`:

- Different colors represent different days
- `.add_legend()` adds a legend **for the hue variable**

6️⃣ Simple mental model 🧠

Component	Role
`FacetGrid (fg)`	Creates the grid structure
`map()`	Applies a plot to each facet
`plt.bar()`	Draws bars for each subset
`hue="day"`	Color grouping
`.add_legend()`	Explains the colors

7️⃣ One-line summary

> **We take `fg` because it represents the faceted plot grid, and we use `map()` because it applies the plotting function (`plt.bar`) to each subset of data defined by the grid (`sex` and `day`).**